

# Towards Self-Driving Networks: Automated What-if Analysis and Synthesis for Dependable Networks

Stefan Schmid (University of Vienna)



# Communication Networks

- **Critical infrastructure** of digital society
  - Popularity of **datacentric applications**: health, business, entertainment, social networking, AI/ML, etc.
  - Evident during ongoing pandemic: online learning, online conferences, etc.
- Traffic is currently growing explosively
  - Especially in, to and, from **datacenters**



Facebook datacenter

Increasingly stringent dependability requirements!

# Requirements vs Reality

Entire countries disconnected...

Data Centre ▶ Networks

**Google routing blunder sent Japan's Internet dark on Friday**

Another big BGP blunder

By Richard Chirgwin 27 Aug 2017 at 22:35

40 □ SHARE ▾

Last Friday, someone in Google fat-thumbed a border gateway protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole.

The trouble began when The Chocolate Factory "leaked" a big route table to Verizon, the result of which was traffic from Japanese giants like NTT and KDDI was sent to Google on the expectation it would be treated as transit.

... 1000s passengers stranded...

**British Airways' latest Total Inability To Support Upwardness of Planes\* caused by Amadeus system outage**

Stuck on the ground awaiting a load sheet? Here's why

By Gareth Corfield 19 Jul 2018 at 11:16

109 □ SHARE ▾



D.A. Baker around the world users recorded as a result of the Amadeus outage

... even 911 services affected!

**Officials: Human error to blame in Minn. 911 outage**

According to a press release, CenturyLink told department of public safety that human error by an employee of a third party vendor was to blame for the outage

Aug 16, 2018

Duluth News Tribune

SAINT PAUL, Minn. — The Minnesota Department of Public Safety Emergency Communication Networks division was told by its 911 provider that an Aug. 1 outage was caused by human error.

Outages simply due to human error! (No attacks...)

# Even Tech-Savvy Companies Struggle



*We discovered a misconfiguration on this pair of switches that caused what's called a “bridge loop” in the network.*

*A network change was [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm.*



*Service outage was due to a series of internal network events that corrupted router data tables.*

*Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems*

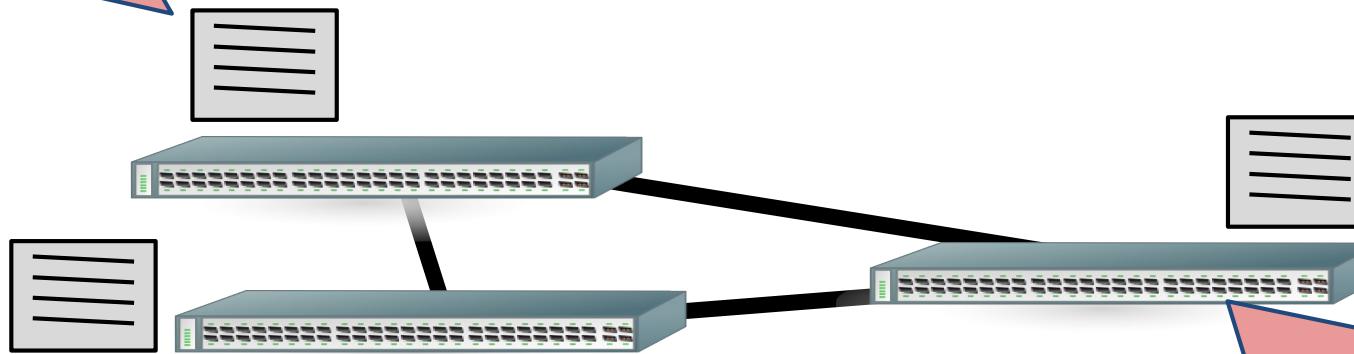


**Also here: due to human errors.**

# No Surprise: Networks Are Complex

Manual, device-centric  
network configurations  
*(CLI, LANmanager)*

Un-evolved best practices  
*(tcpdump, traceroute - from the 1990s)*

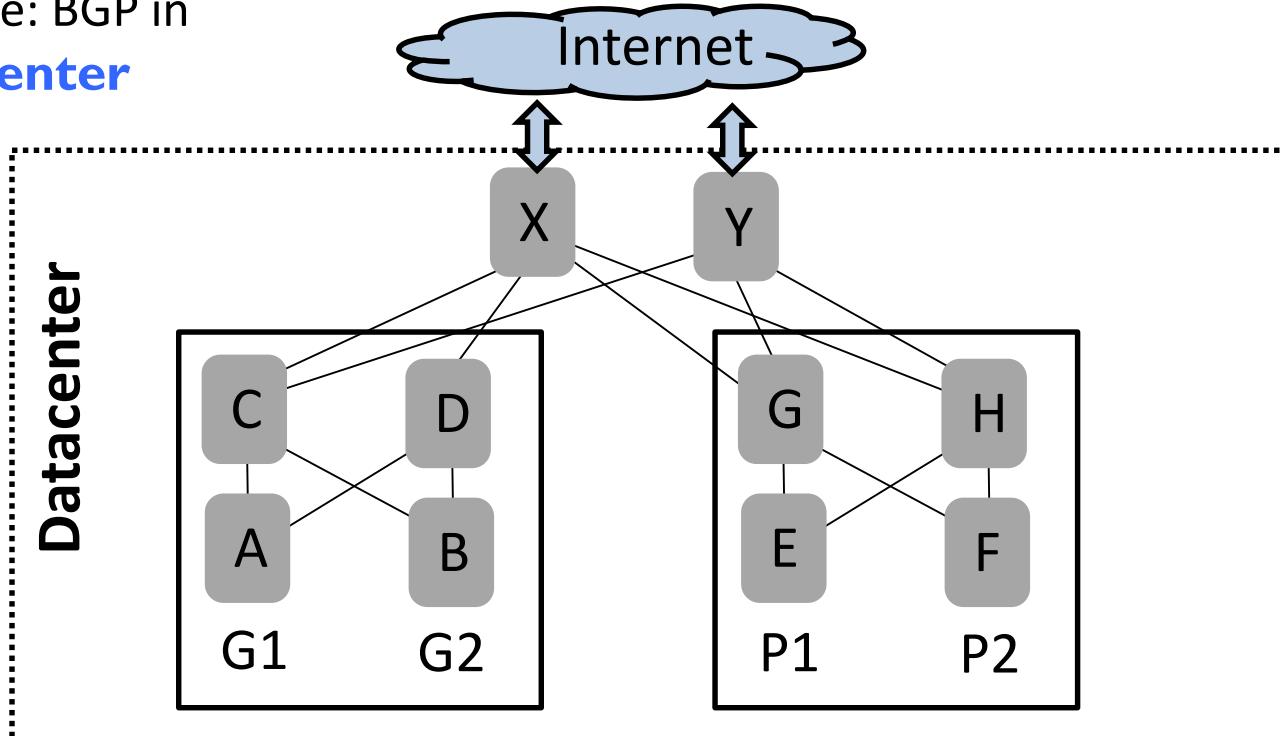


500-router network: typically  
**>1 million lines** of configuration

Complex, leaky, low-level interfaces  
*(VLANs, Spanning Tree, Routing)*

# Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

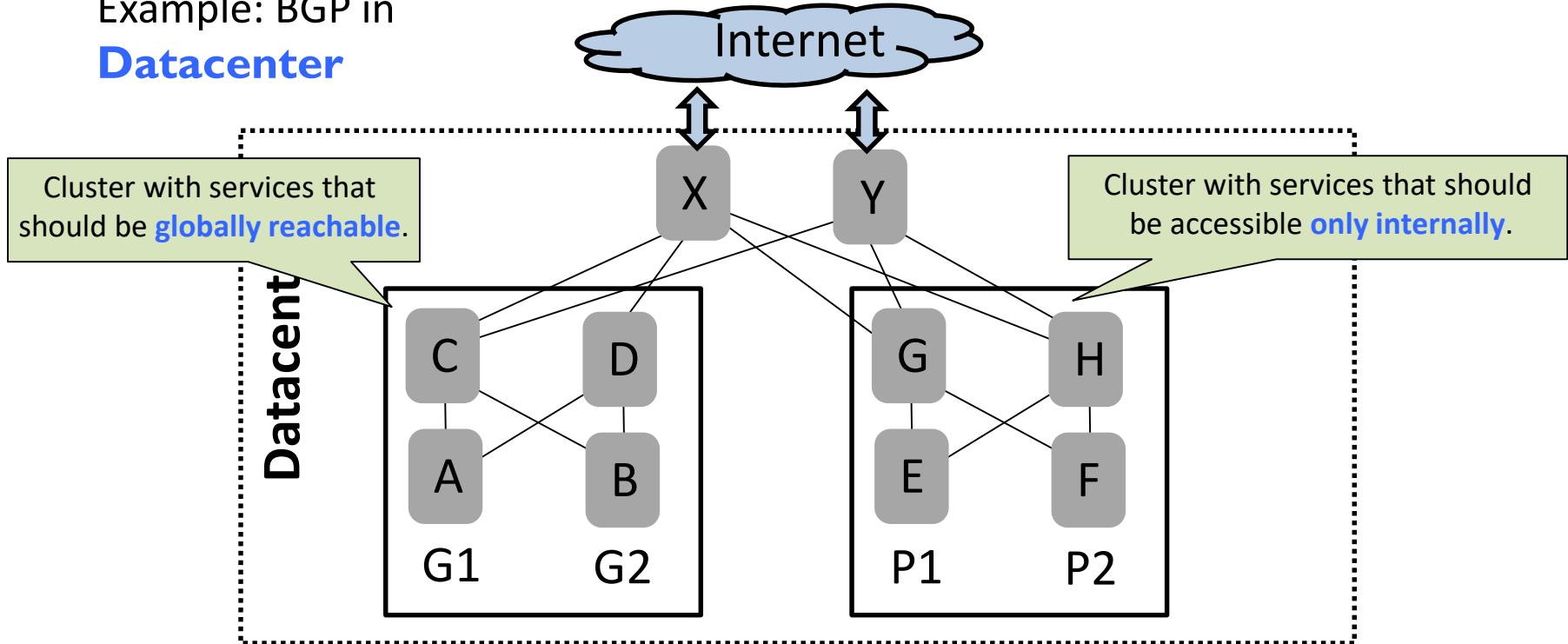
Example: BGP in  
**Datacenter**



Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

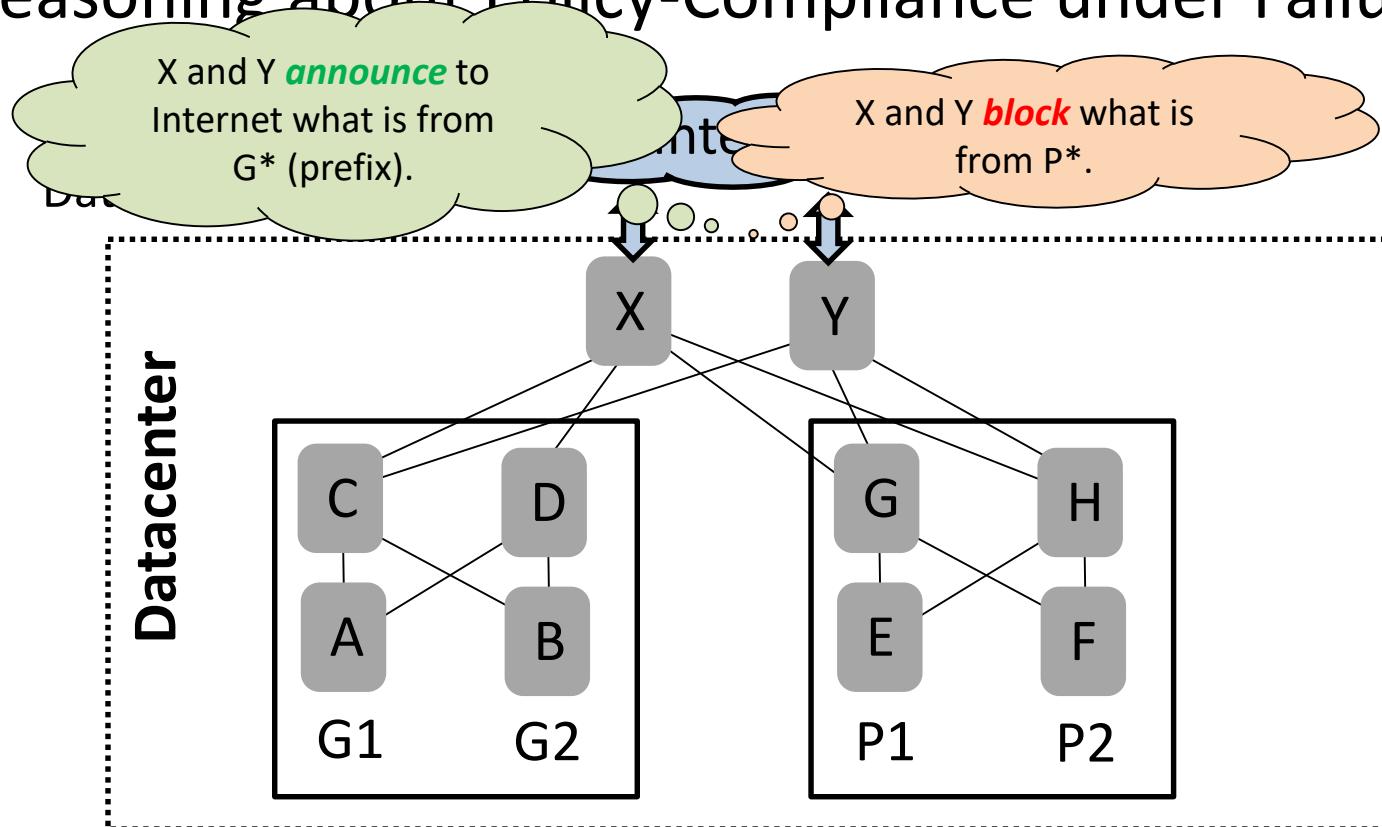
# Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

Example: BGP in  
**Datacenter**



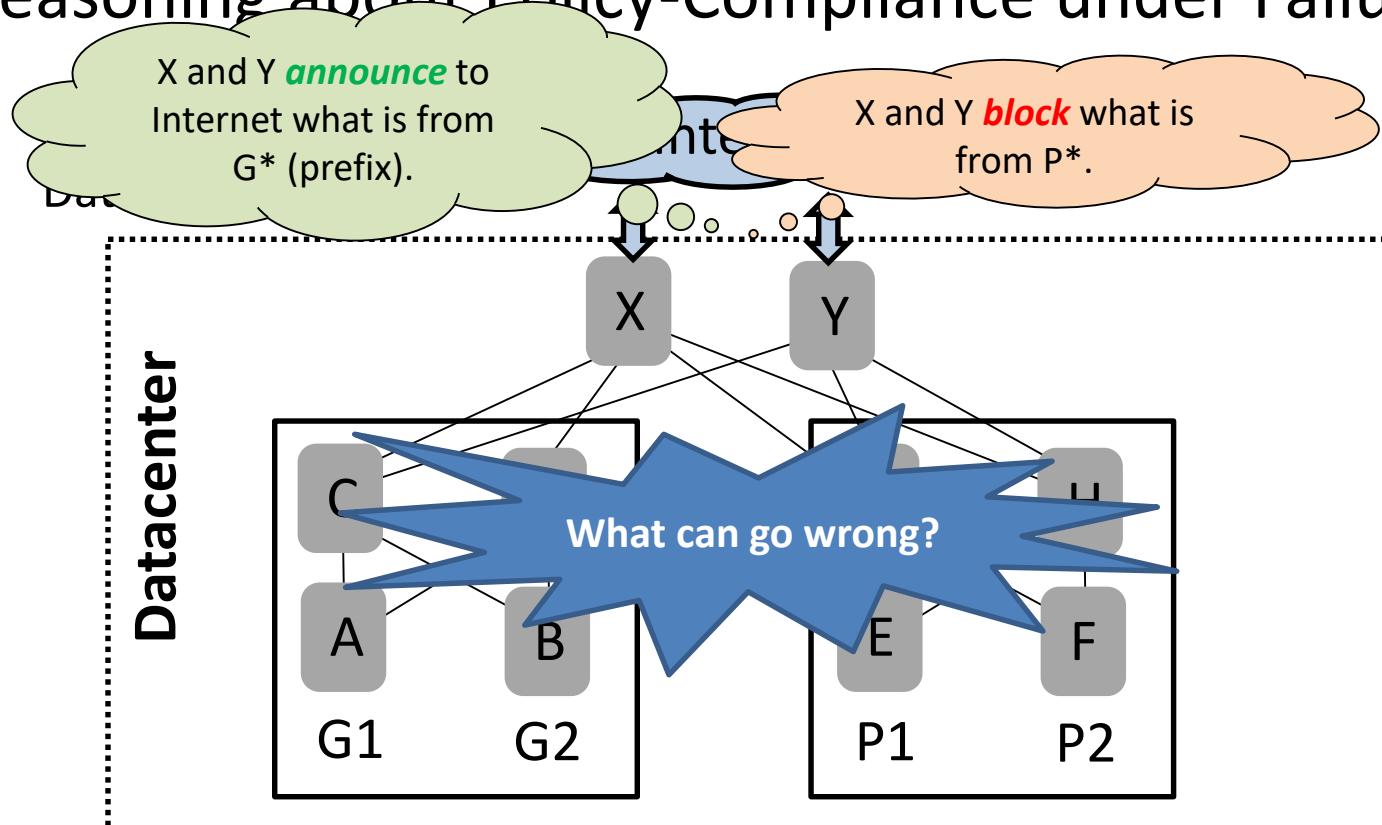
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

# Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



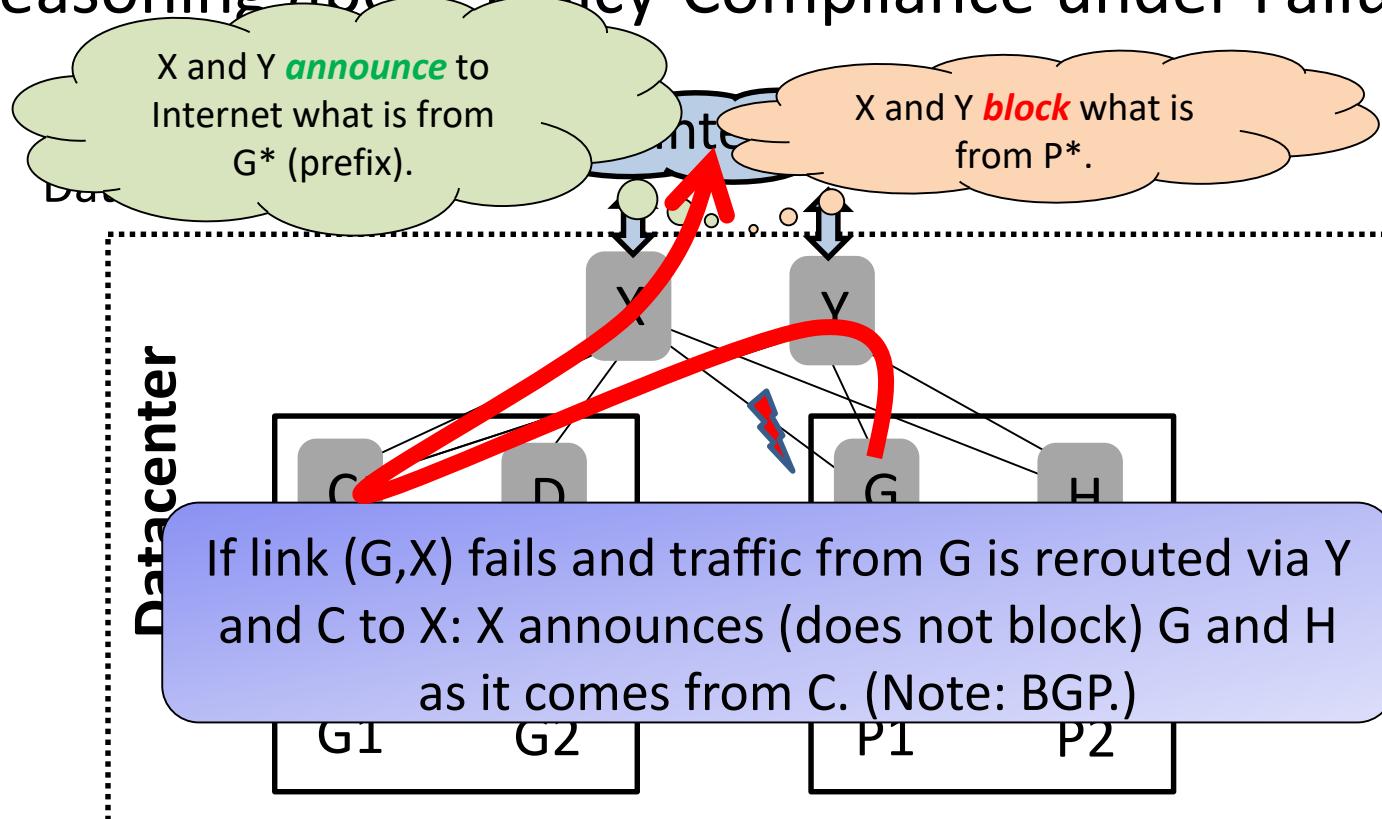
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

# Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

# Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

# We're Falling Behind the Curve: Increasing Complexity, Software from the 90s

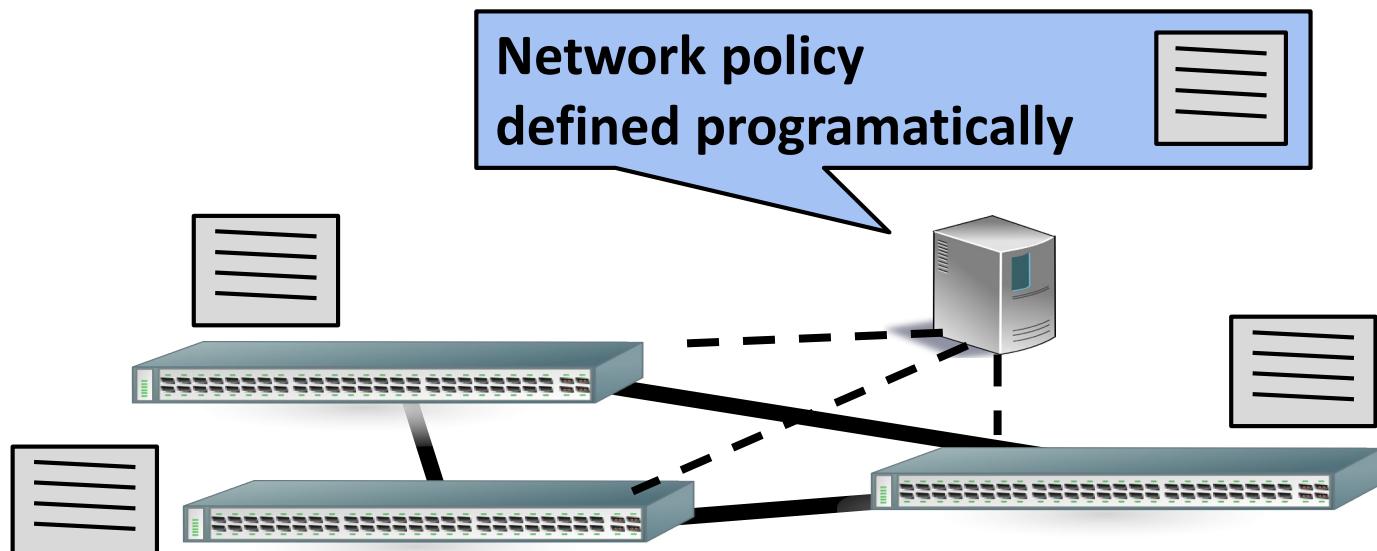
- Anecdote **Wall Street bank**: outage of a datacenter
  - Lost revenue measured in **1 mio\$/min**
- Quickly, an emergency team was assembled with experts in compute, storage and networking:
  - **The compute team:** *reams of logs*, written experiments to reproduce and *isolate the error*
  - **The storage team:** *system logs* were affected, *workaround programs*.
  - “All the **networking team** had were *two tools invented over twenty years ago* to merely test end-to-end connectivity. Neither tool could reveal *problems with the switches*, the *congestion* experienced.”



Source: «The world's fastest and most programmable networks»  
White Paper Barefoot Networks

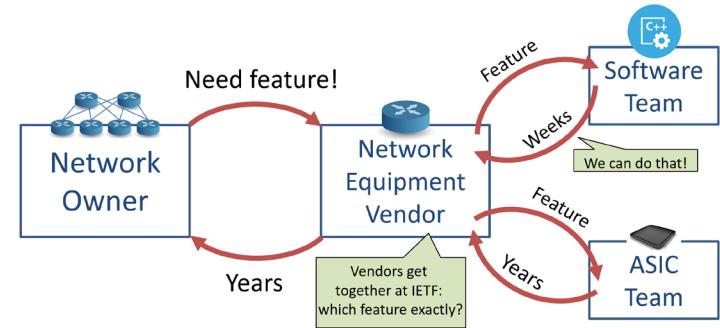
# There is Hope: Software-Defined Networks

- **Automation** and abstraction
- Directly program routing behavior (i.e., push forwarding tables)
- Open interfaces: „the **Linux** of networking“



# Remark (for the Network Experts...)

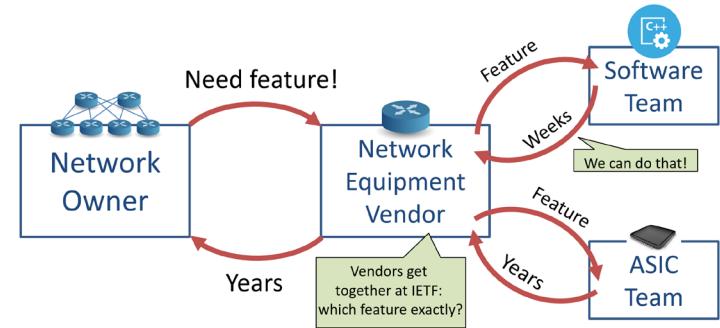
- Networks currently become *programmable* in the control plane and the data plane
  - **Control plane**: network-wide algorithms (e.g., routing)
  - **Data plane**: router/switch level algorithms (e.g., forwarding, filtering)
- Motivation in both cases: software usually trumps hardware in terms of *innovation speed*
- **Software can be fast:**
  - Our Tofino switch: operates at *6.5 Tb/s*
  - Order of magnitude faster than our faculty's Internet connection: can switch entire *Netflix catalogue* in 20sec
  - While *running a 4000 line program* on any packet...
  - .. and not being more costly or consume more power



Example: VxLAN

# Remark (for the Network Experts...)

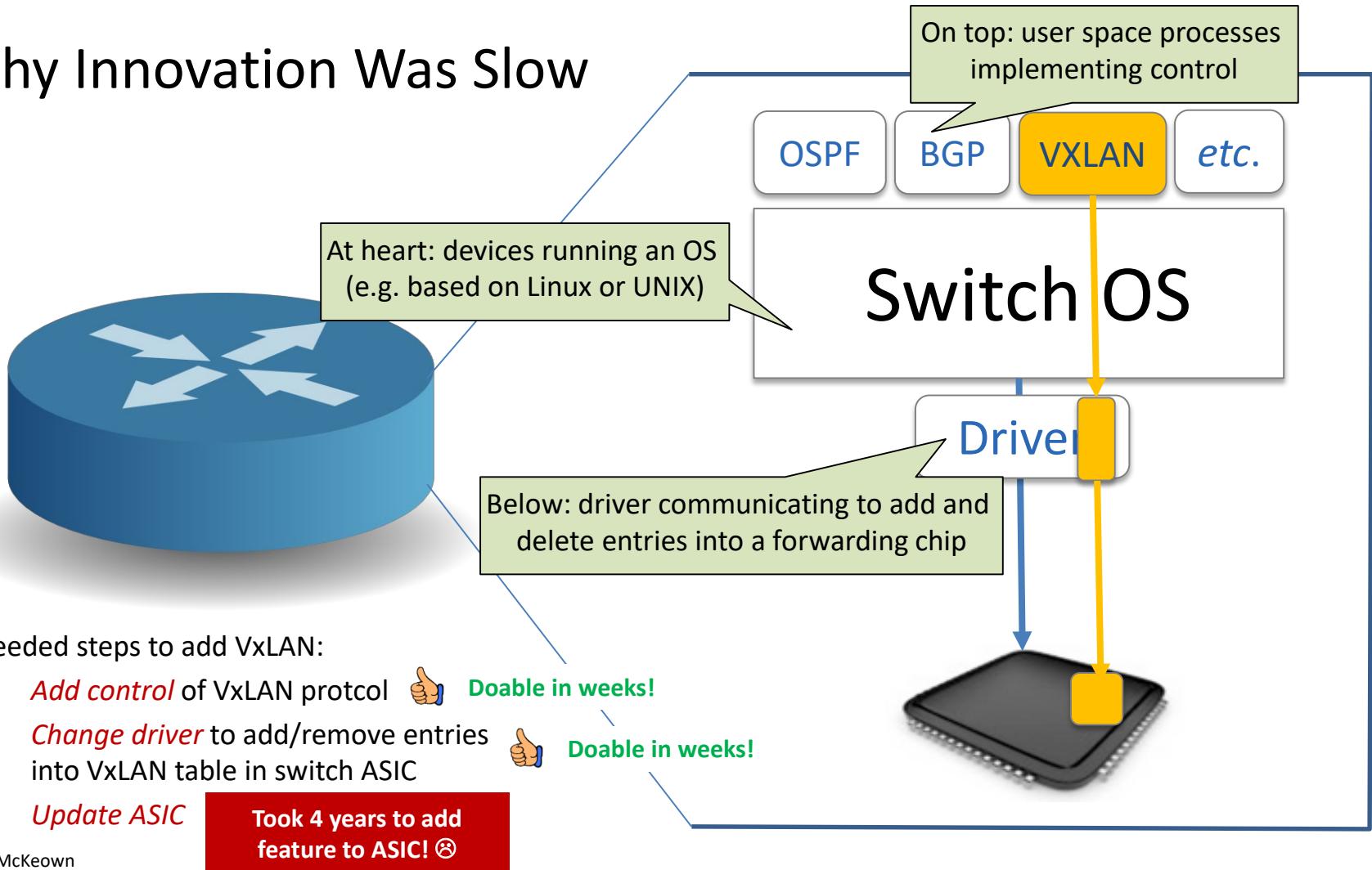
- Networks currently become *programmable* in the control plane and the data plane
  - **Control plane**: network-wide algorithms (e.g., routing)
  - **Data plane**: router/switch level algorithms (e.g., forwarding, filtering)
- Motivation in both cases: software usually trumps hardware in terms of *innovation speed*
- **Software can be fast:**
  - Our Tofino switch: operates at *6.5 Tb/s*
  - Order of magnitude faster than our faculty's Internet connection: can switch entire *Netflix catalogue* in 20sec
  - While *running a 4000 line program* on any packet...
  - .. and not being more costly or consume more power



Example: VxLAN

... and: the automation trend is not limited to SDN.

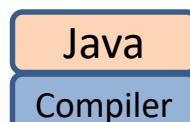
# Why Innovation Was Slow



# Now Networking is Catching Up

Similar to other IT trends: can now *write high-level program* and compile it to *domain specific processor*.

Computers



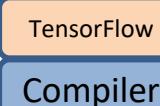
Graphics



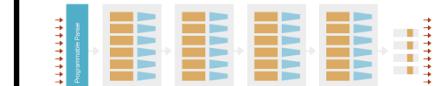
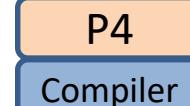
Signal Processing



Machine Learning



Networking



PISA/Tofino

# Roadmap

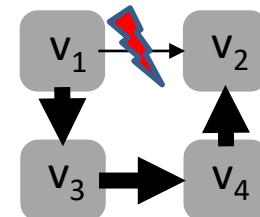
- **A Static Problem:** Policy Compliance  
*Under Failures*
  - AalWiNes: Fast Automated **What-if Analysis** for Networks (INFOCOM 2018, ACM CoNEXT 2018, ACM CoNEXT 2019, TACAS 2021)
- **A Dynamic Problem:** **Scheduling** Consistent Network *Updates*
  - Latte and quantitative extensions (PODC 2015, ICALP 2018, PERFORMANCE 2021)



# Background: Rerouting Under Failures

Two approaches to react to link failures

- In the **control plane**: just re-invoke (shortest path) routing protocol
  - Always re-establishes connectivity but slow
- In the **data plane**: pre-defined local failover rules
  - Orders of magnitude *faster* Our focus!

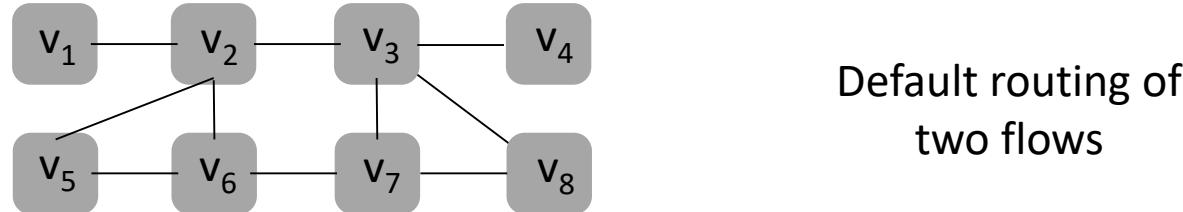


Restoration in control plane takes time -> packet drops!



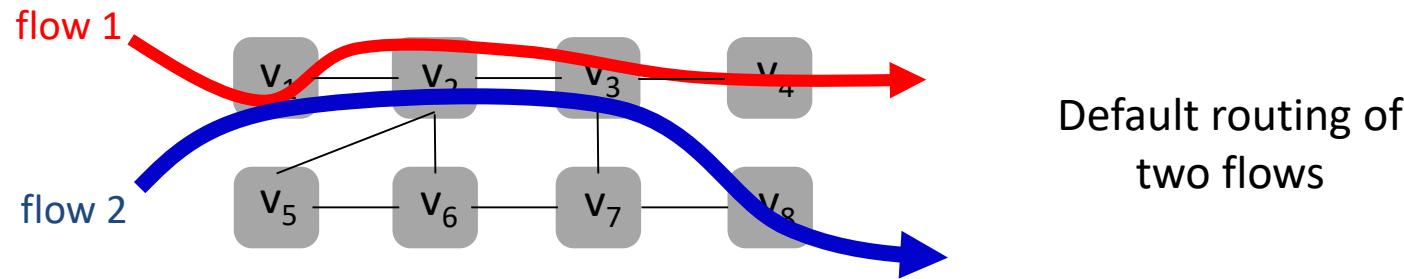
# How (MPLS) Networks Work

- Forwarding based on **top label** of label **stack**



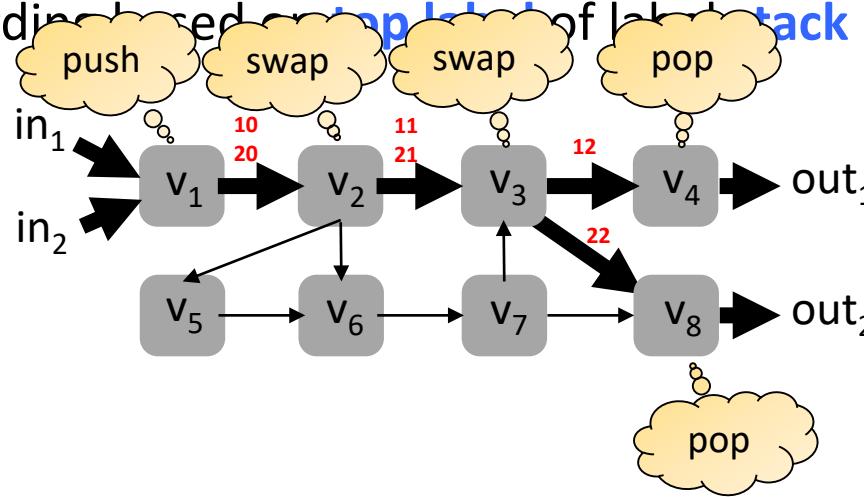
# How (MPLS) Networks Work

- Forwarding based on **top label** of label **stack**



# How (MPLS) Networks Work

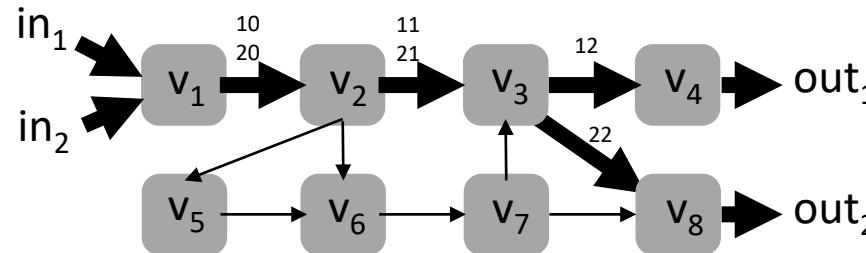
- Forwarding based on top **left** of label stack



Default routing of  
two flows

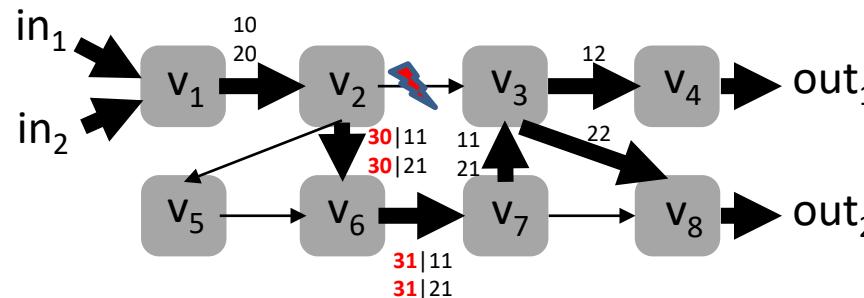
# Fast Reroute Around 1 Failure

- Forwarding based on **top label** of label **stack** (in packet header)



Default routing of  
two flows

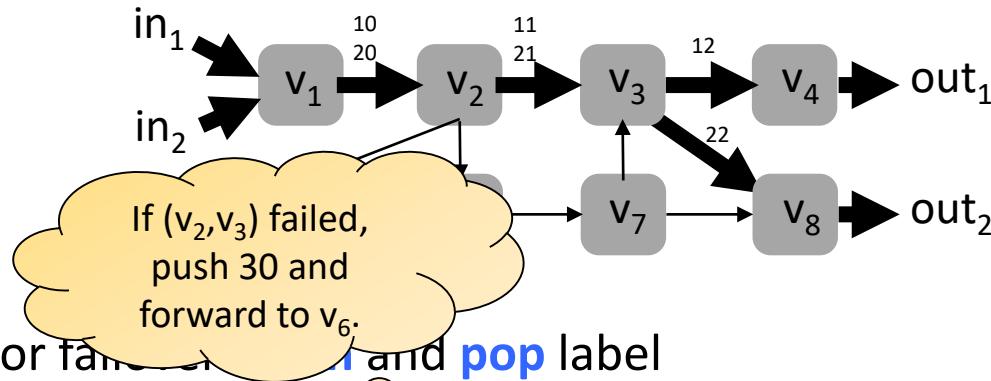
- For failover: **push** and **pop** label



One failure: **push** 30:  
route around ( $v_2, v_3$ )

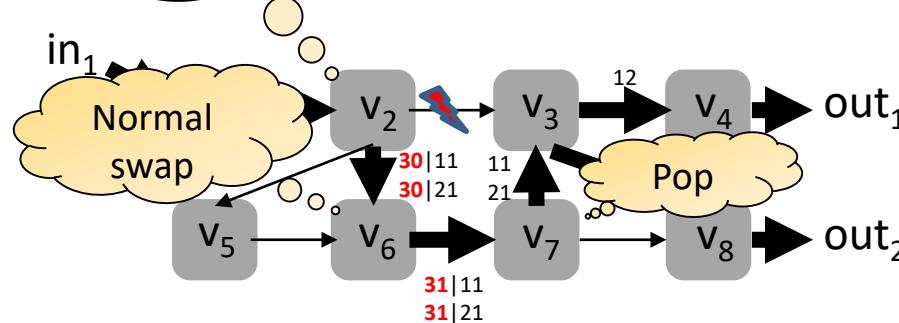
# Fast Reroute Around 1 Failure

- Forwarding based on **top label** of label **stack** (in packet header)



Default routing of  
two flows

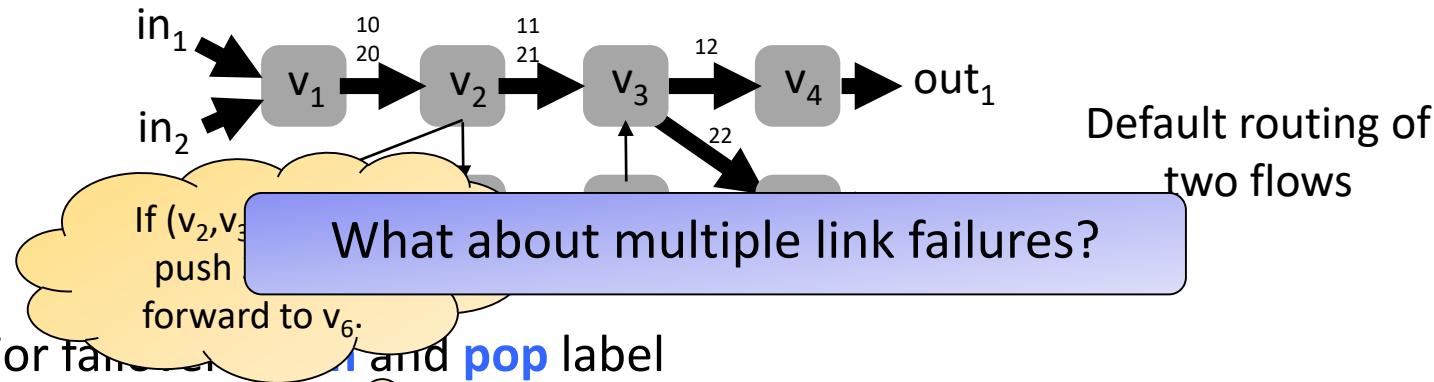
- For failure, **swap** and **pop** label



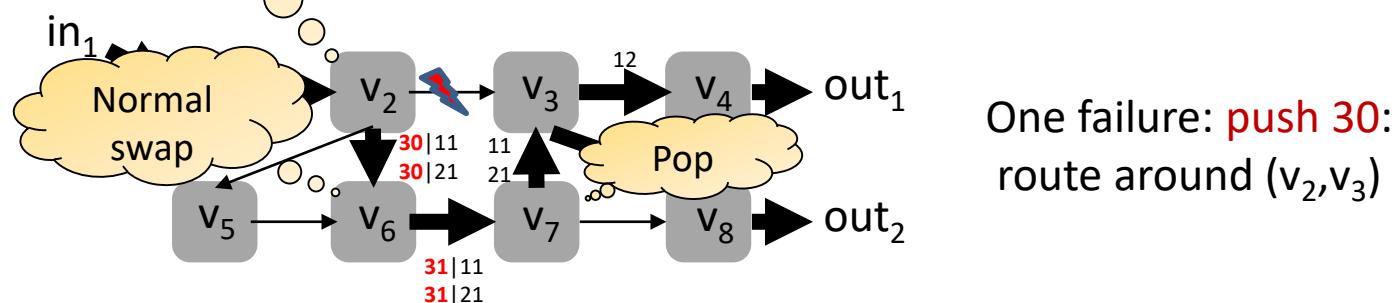
One failure: **push 30:**  
route around  $(v_2, v_3)$

# Fast Reroute Around 1 Failure

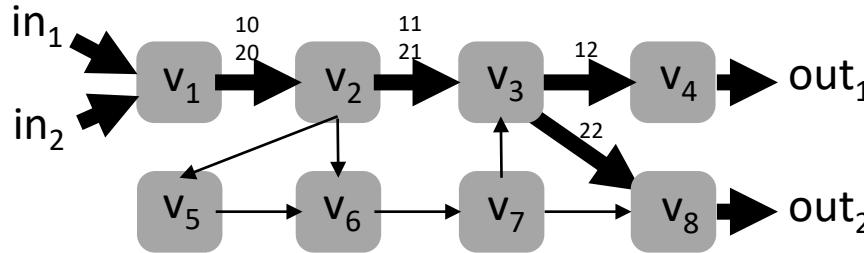
- Forwarding based on **top label** of label **stack** (in packet header)



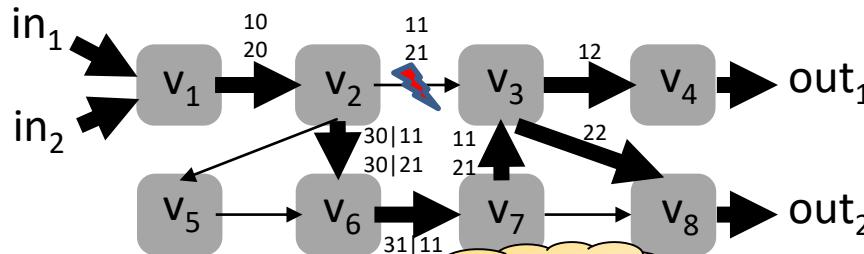
- For failure, **push** and **pop** label



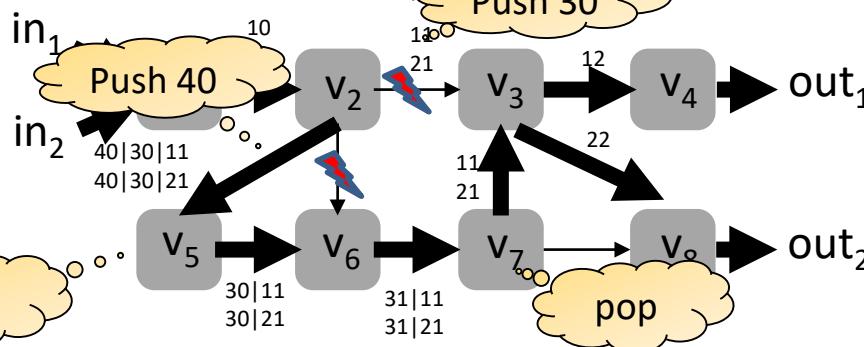
# 2 Failures: Push *Recursively*



Original Routing



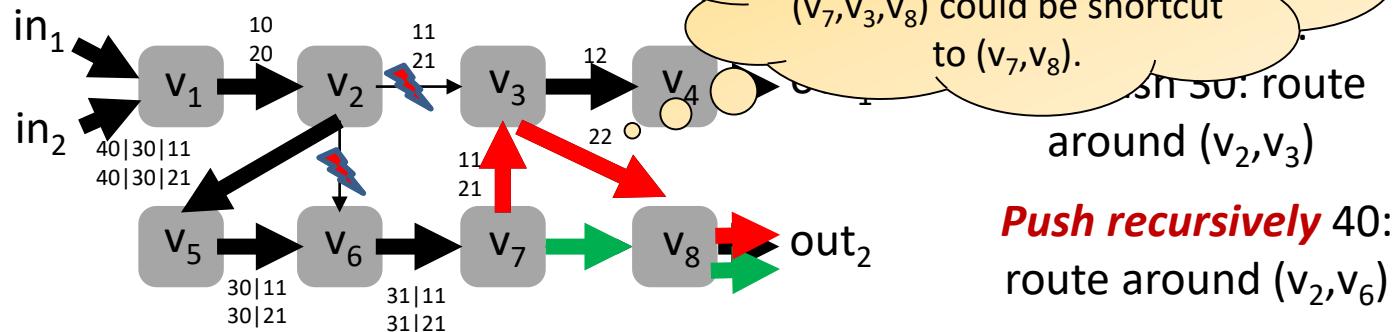
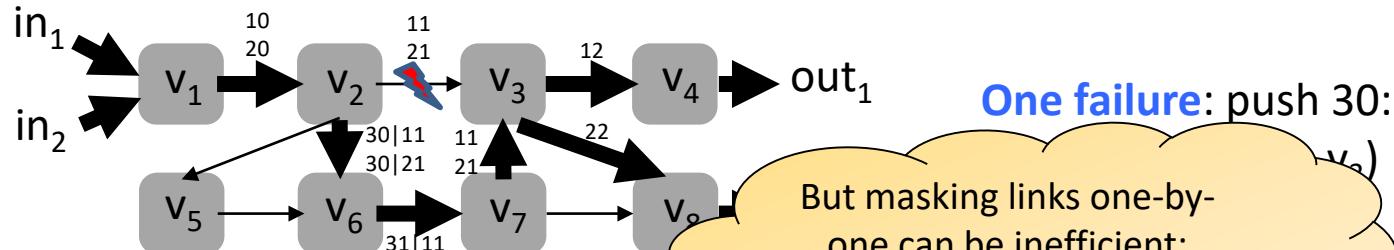
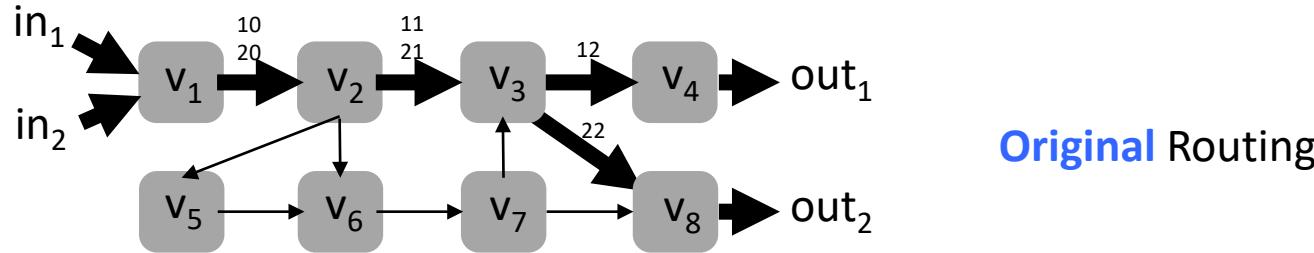
One failure: push 30:  
route around ( $v_2, v_3$ )



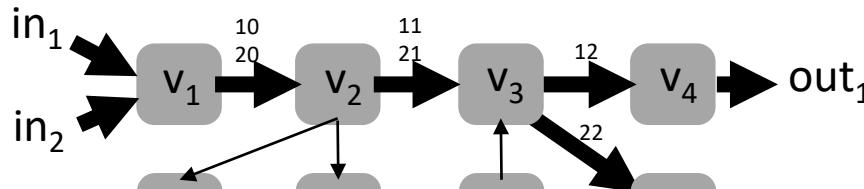
Two failures:  
first push 30: route  
around ( $v_2, v_3$ )

*Push recursively* 40:  
route around ( $v_2, v_6$ )

# 2 Failures: Push *Recursively*

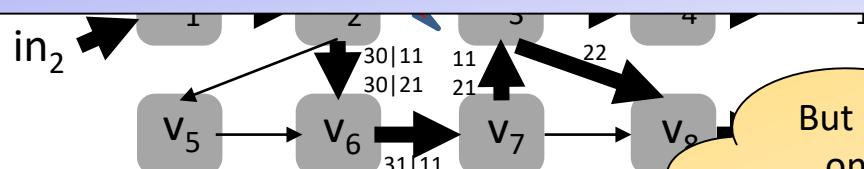


# 2 Failures: Push *Recursively*



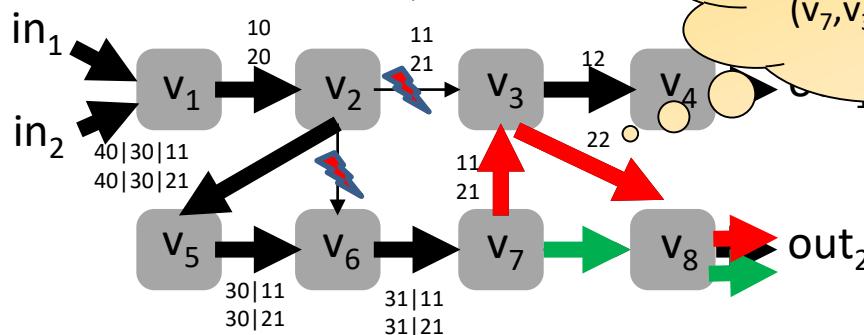
Original Routing

More efficient but also more complex:  
Cisco does **not recommend** using this option!



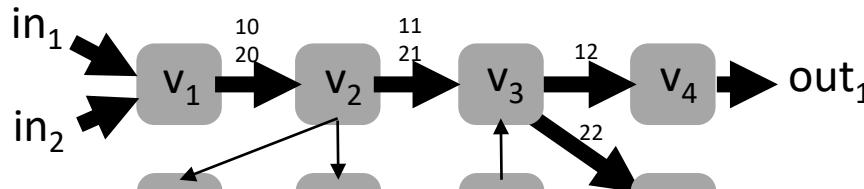
One failure: push 30:

But masking links one-by-one can be inefficient:  
 $(v_7, v_3, v_8)$  could be shortcut  
to  $(v_7, v_8)$ .



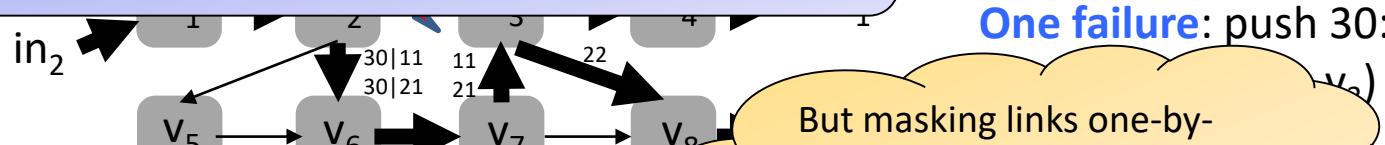
Push recursively 40:  
route around  $(v_2, v_6)$

# 2 Failures: Push *Recursively*



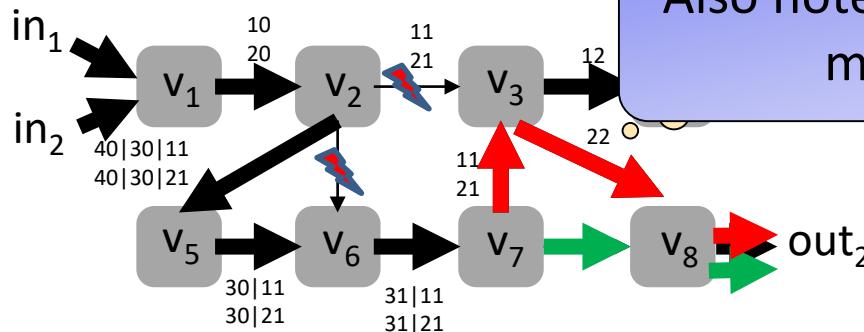
Original Routing

More efficient but also more complex:  
Cisco does **not recommend** using this option!



One failure: push 30:

But masking links one-by-



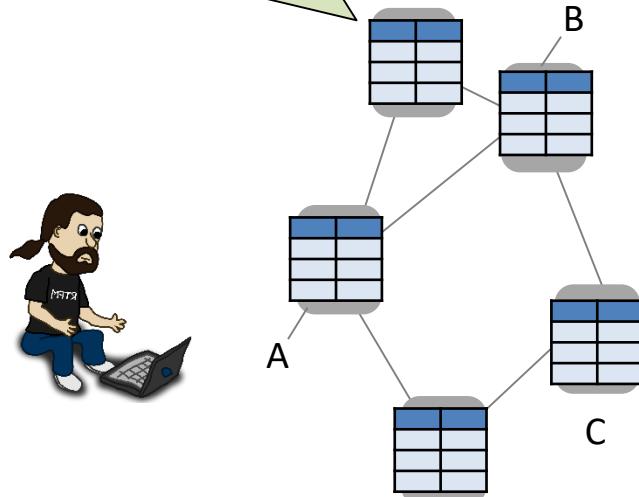
Also note: due to push, **header size** may grow arbitrarily!

around  $(v_2, v_6)$

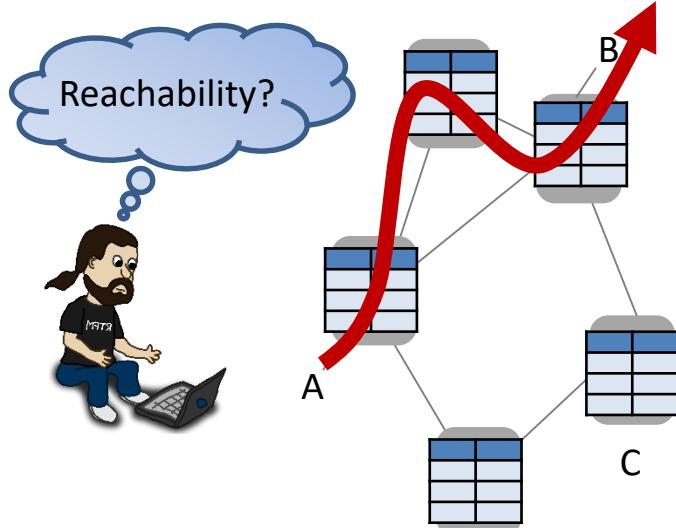
**Push recursively 40:**  
route around  $(v_2, v_6)$

# Responsibilities of a Sysadmin

Routers and switches store list of **forwarding rules**, and conditional **failover rules**.



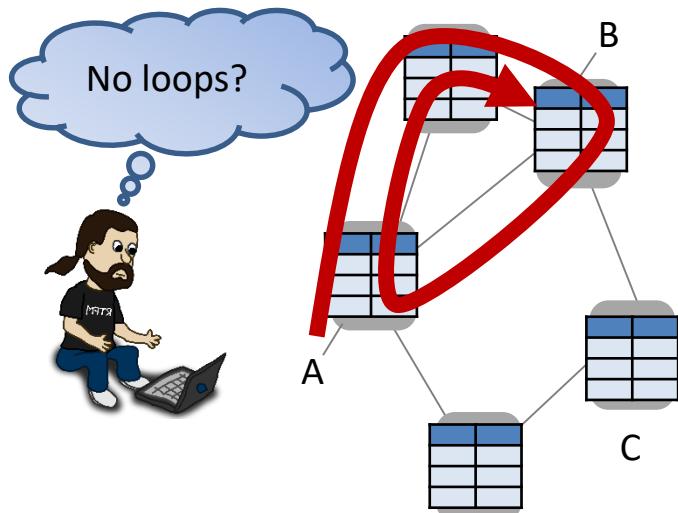
# Responsibilities of a Sysadmin



**Sysadmin** responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?

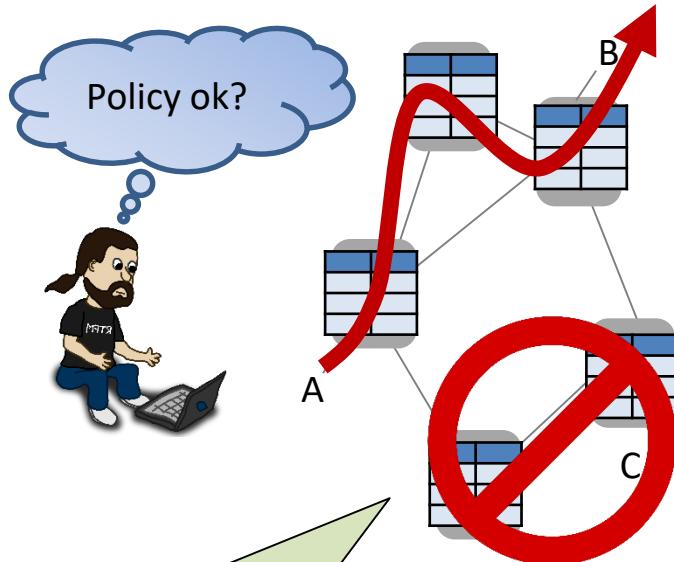
# Responsibilities of a Sysadmin



**Sysadmin** responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?

# Responsibilities of a Sysadmin

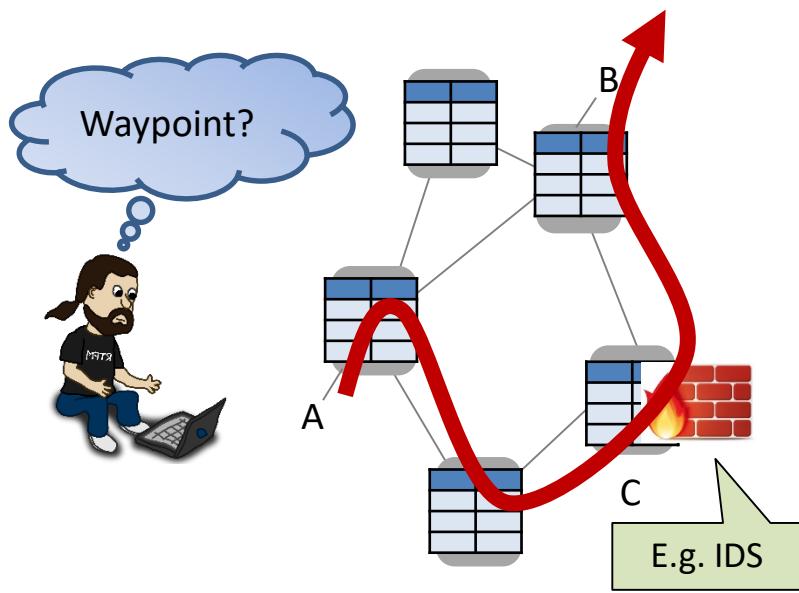


E.g. **NORDUnet**: no traffic via Iceland (expensive!).

**Sysadmin** responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?

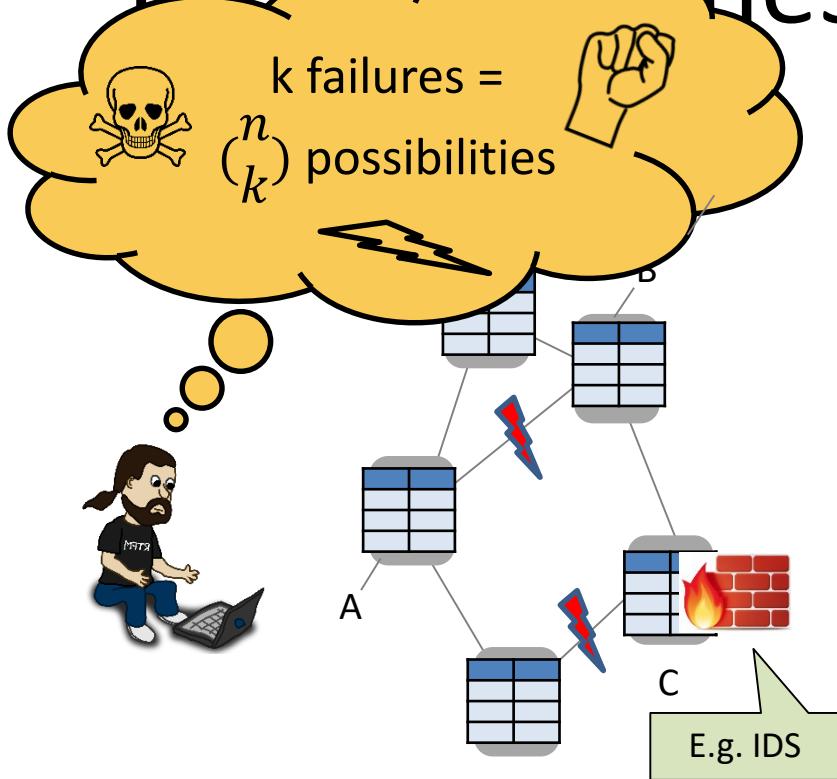
# Responsibilities of a Sysadmin



**Sysadmin** responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed via a node C (e.g., intrusion detection system or a firewall)?

# Responsibilities of a Sysadmin

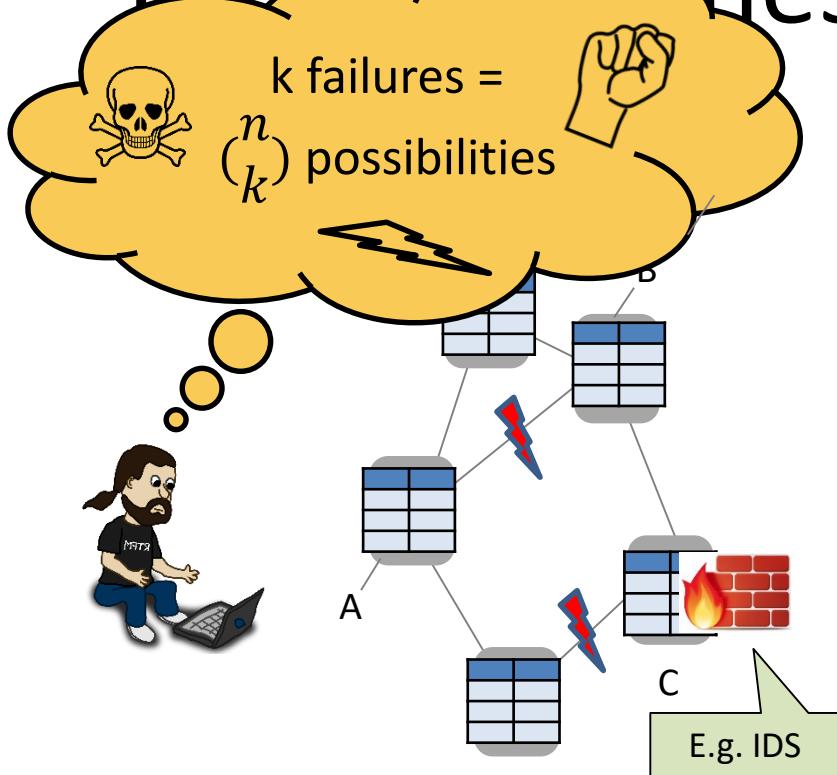


Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed via a node C (e.g., intrusion detection system or a firewall)?

*... and everything even under multiple failures?!*

# Responsibilities of a Sysadmin



Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed via a node C (e.g., intrusion detection system or a firewall)?

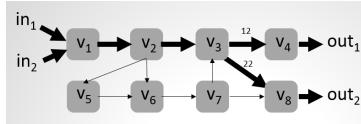
*... and everything even under multiple failures?!*

Generalization: service chaining!

# Approach: Automation and Formal Methods

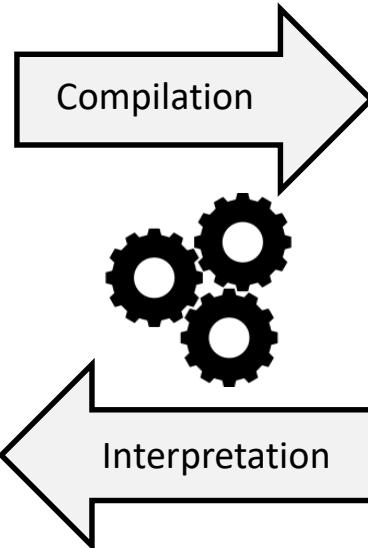


FT	In-I	In-Label	Out-I	op
$\tau_{v_1}$	$in_1$	$\perp$	$(v_1, v_2)$	$push(10)$
$\tau_{v_2}$	$(v_1, v_2)$	10	$(v_2, v_3)$	$push(20)$
$\tau_{v_3}$	$(v_1, v_2)$	20	$(v_2, v_3)$	$swap(11)$
$\tau_{v_4}$	$(v_2, v_3)$	11	$(v_3, v_4)$	$swap(12)$
$\tau_{v_5}$	$(v_2, v_3)$	21	$(v_3, v_4)$	$swap(22)$
$\tau_{v_6}$	$(v_3, v_4)$	11	$(v_3, v_5)$	$swap(12)$
$\tau_{v_7}$	$(v_3, v_4)$	21	$(v_3, v_5)$	$swap(22)$
$\tau_{v_8}$	$(v_2, v_5)$	40	$(v_5, v_6)$	$pop$
$\tau_{v_9}$	$(v_2, v_6)$	30	$(v_6, v_7)$	$swap(31)$
$\tau_{v_{10}}$	$(v_5, v_6)$	30	$(v_6, v_7)$	$swap(31)$
$\tau_{v_{11}}$	$(v_5, v_6)$	61	$(v_6, v_7)$	$swap(62)$
$\tau_{v_{12}}$	$(v_5, v_6)$	71	$(v_6, v_7)$	$swap(72)$
$\tau_{v_{13}}$	$(v_6, v_7)$	31	$(v_7, v_8)$	$pop$
$\tau_{v_{14}}$	$(v_6, v_7)$	62	$(v_7, v_8)$	$swap(11)$
$\tau_{v_{15}}$	$(v_6, v_7)$	72	$(v_7, v_8)$	$swap(22)$
$\tau_{v_{16}}$	$(v_3, v_8)$	22	$out_1$	$pop$
$\tau_{v_{17}}$	$(v_3, v_8)$	22	$out_2$	$pop$



local FFT	Out-I	In-Label	Out-I	op
$\tau_{v_2}$	$(v_2, v_3)$	11	$(v_2, v_6)$	$push(30)$
	$(v_2, v_3)$	21	$(v_2, v_6)$	$push(30)$
	$(v_2, v_6)$	30	$(v_2, v_5)$	$push(40)$
global FFT	Out-I	In-Label	Out-I	op
$\tau'_{v_2}$	$(v_2, v_3)$	11	$(v_2, v_6)$	$swap(61)$
	$(v_2, v_3)$	21	$(v_2, v_6)$	$swap(71)$
	$(v_2, v_6)$	61	$(v_2, v_5)$	$push(40)$
	$(v_2, v_6)$	71	$(v_2, v_5)$	$push(40)$

Router configurations  
(Cisco, Juniper, etc.)



$$pX \Rightarrow qXX$$

$$pX \Rightarrow qYX$$

$$qY \Rightarrow rYY$$

$$rY \Rightarrow r$$

$$rX \Rightarrow pX$$

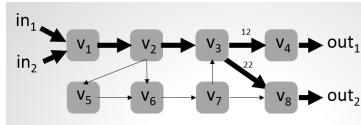
Pushdown Automaton and  
Prefix Rewriting Systems

# Approach: Automation

Use cases: Sysadmin **issues queries** to test certain properties, or do it on a **regular basis** automatically!



FT	In-I	In-Label	Out-I	op
$\tau_{v_1}$	$in_1$	$\perp$	$(v_1, v_2)$	$push(10)$
$\tau_{v_2}$	$(v_1, v_2)$	10	$(v_2, v_3)$	$swap(11)$
$\tau_{v_3}$	$(v_1, v_2)$	20	$(v_2, v_3)$	$swap(21)$
$\tau_{v_4}$	$(v_2, v_3)$	11	$(v_3, v_4)$	$swap(12)$
$\tau_{v_5}$	$(v_2, v_3)$	21	$(v_3, v_4)$	$swap(22)$
$\tau_{v_6}$	$(v_3, v_4)$	11	$(v_3, v_5)$	$swap(12)$
$\tau_{v_7}$	$(v_3, v_4)$	21	$(v_3, v_5)$	$swap(22)$
$\tau_{v_8}$	$(v_3, v_4)$	12	$out_1$	$pop$
$\tau_{v_9}$	$(v_2, v_5)$	40	$(v_5, v_6)$	$pop$
$\tau_{v_{10}}$	$(v_2, v_6)$	30	$(v_6, v_7)$	$swap(31)$
$\tau_{v_{11}}$	$(v_5, v_6)$	30	$(v_6, v_7)$	$swap(31)$
$\tau_{v_{12}}$	$(v_5, v_6)$	61	$(v_6, v_7)$	$swap(62)$
$\tau_{v_{13}}$	$(v_5, v_6)$	71	$(v_6, v_7)$	$swap(72)$
$\tau_{v_{14}}$	$(v_6, v_7)$	31	$(v_6, v_8)$	$pop$
$\tau_{v_{15}}$	$(v_6, v_7)$	62	$(v_7, v_8)$	$swap(11)$
$\tau_{v_{16}}$	$(v_6, v_7)$	72	$(v_7, v_8)$	$swap(22)$
$\tau_{v_{17}}$	$(v_3, v_8)$	22	$out_2$	$pop$
$\tau_{v_{18}}$	$(v_7, v_8)$	22	$out_2$	$pop$



local FFT	Out-I	In-Label	Out-I	op
$\tau_{v_2}$	$(v_2, v_3)$	11	$(v_2, v_6)$	$push(30)$
	$(v_2, v_3)$	21	$(v_2, v_6)$	$push(30)$
	$(v_2, v_6)$	30	$(v_2, v_5)$	$push(40)$
global FFT	Out-I	In-Label	Out-I	op
$\tau'_{v_2}$	$(v_2, v_3)$	11	$(v_2, v_6)$	$swap(61)$
	$(v_2, v_3)$	21	$(v_2, v_6)$	$swap(71)$
	$(v_2, v_6)$	61	$(v_2, v_5)$	$push(40)$
	$(v_2, v_6)$	71	$(v_2, v_5)$	$push(40)$

Router **configurations**  
(Cisco, Juniper, etc.)

Compilation



$pX \Rightarrow qXX$

$pX \Rightarrow qYX$

$qY \Rightarrow rYY$

$rY \Rightarrow r$

$rX \Rightarrow pX$

Interpretation

Pushdown Automaton and  
Prefix Rewriting Systems

# AalWiNes

**AalWiNes** Indian Ocean

MPLS Reachability Analysis & Visualization Tool

Model *Aarnet* +

Query `<ip> [#Sydney1] .* [Brisbane2#.] <ip> 0` -

Examples:  
`<ip> [#Sydney1] .* [Brisbane2#.] <ip> 0  
<smpls ip> [#Sydney1] .* [Brisbane2#.] <mpls* smpls ip> 1`

Initial header: `ip`  
Route restriction: `[#.Sydney1] .* [Brisbane2#.]`  
Final header: `ip`  
Max link failures: `0`  
...

Options +

Run Validation Run Validation

**Result Satisfied** -

Query: `<ip> [#Sydney1] .* [Brisbane2#.] <ip> 0`

```
<ip6> : [#Sydney1]
  push($43)
<s43,ip6> : [Sydney1#Brisbane1]
  swap($44)
<s44,ip6> : [Brisbane1#Brisbane2]
  pop()
<ip6> : [Brisbane2#]
```

The map displays several blue curved lines representing MPLS reachability paths between Sydney and Brisbane, passing through various cities like Melbourne, Adelaide, and Perth. The map also shows parts of Indonesia, the Philippines, and surrounding island nations.

**About AalWiNes**

A tool for MPLS reachability analysis and visualization from:

- Aalborg University Department of Computer Science
- University of Vienna Communication Technologies Group

Have a look at the [Tool Website](#) & [Tool and query language documentation](#)

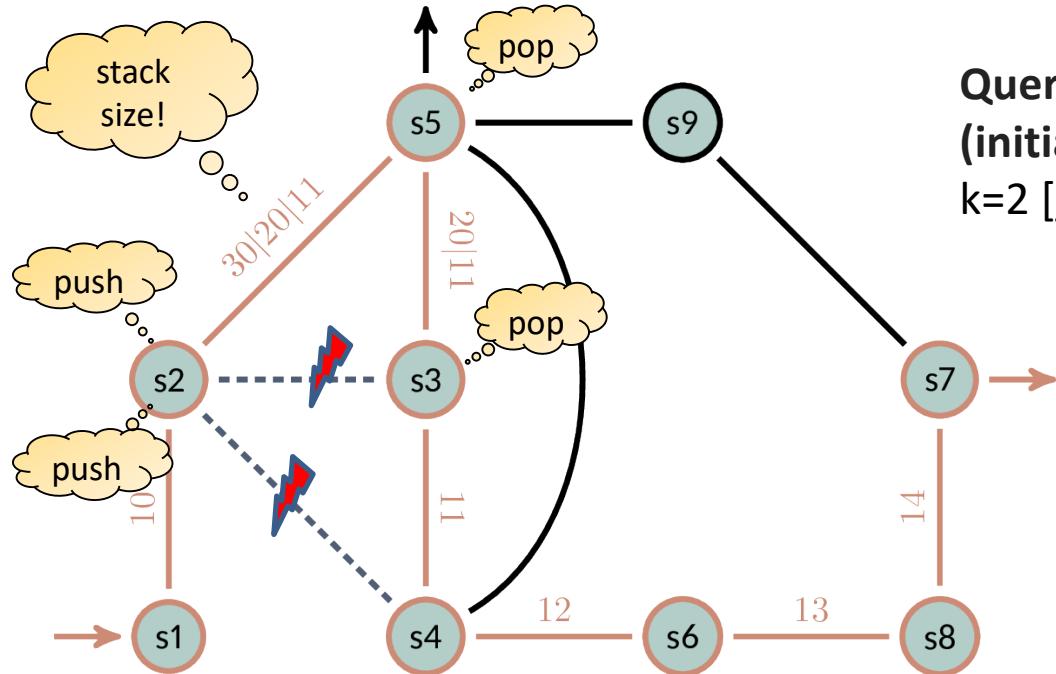
Witness

Dozens of networks

Online demo: <https://demo.aalwines.cs.aau.dk/>  
Source code: <https://github.com/DEIS-Tools/AalWiNes>

# Example

Can traffic starting with [] go through  $s_5$ , under up to  $k=2$  failures?



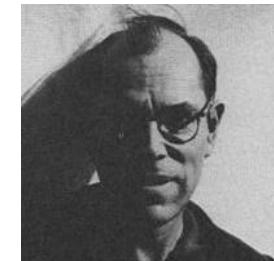
Query: 3 regular expressions  
(initial and final header, route)  
 $k=2$   $[] \xrightarrow{s_1} s_5 \xrightarrow{s_7} []$

2 failures

YES  
(Polynomial time!)

# Why AalWiNes is Fast (Polytime): Automata Theory

- For fast verification, we can use the result by **Büchi**: the set of all reachable configurations of a pushdown automaton  $a$  is **regular set**
- We hence simply use **Nondeterministic Finite Automata (NFAs)** when reasoning about the pushdown automata
- The resulting **regular operations** are all **polynomial time**



Julius Richard Büchi  
1924-1984  
Swiss logician

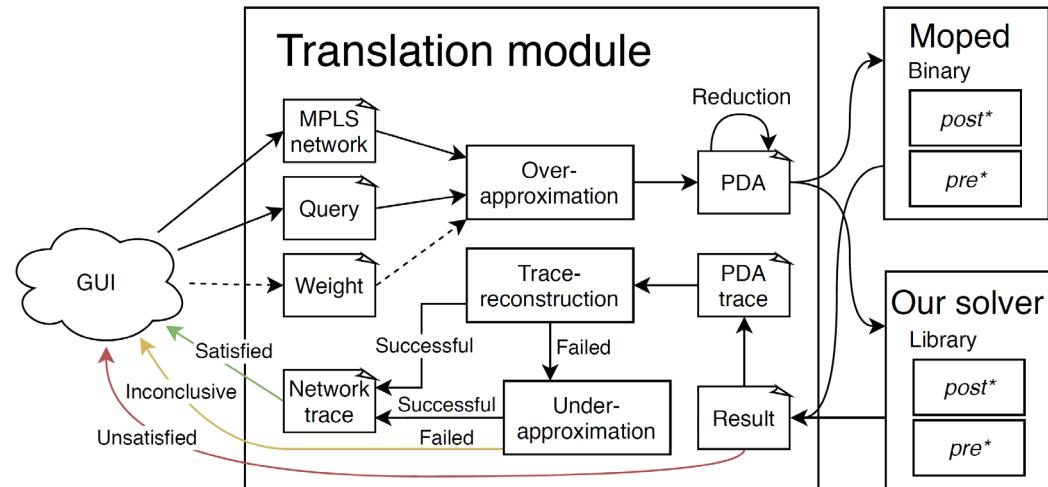
# AalWiNes

**Part 1:** Parses query  
and constructs Push-  
Down System (PDS)

- In Python 3

**Part 2:** Reachability  
analysis of  
constructed PDS

- Using **Moped** tool



Resp. our new weighted extension and  
much faster implementation in C++.

# Network Model

- Network: a 7-tuple

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$

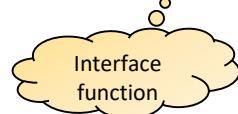
The diagram illustrates the components of a network model tuple. It consists of five yellow thought bubbles connected by dotted lines to the corresponding elements in the tuple definition:

- A bubble labeled "Links" connects to the element  $I_v^{in}$ .
- A bubble labeled "Outgoing interfaces" connects to the element  $I_v^{out}$ .
- A bubble labeled "Nodes" connects to the element  $V$ .
- A bubble labeled "Incoming interfaces" connects to the element  $E$ .
- A bubble labeled "Set of labels in packet header" connects to the element  $\delta_v^F$ .

# Network Model

- Network: a 7-tuple

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$



**Interface function:** maps outgoing interface to next hop node and incoming interface to previous hop node

$$\lambda_v : I_v^{in} \cup I_v^{out} \rightarrow V$$

That is:  $(\lambda_v(in), v) \in E$  and  $(v, \lambda_v(out)) \in E$

# Network Model

- Network: a 7-tuple

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$



**Routing function:** for each set of **failed links**  $F \subseteq E$ , the routing function

$$\delta_v^F : I_v^{in} \times L^* \rightarrow 2^{(I_v^{out} \times L^*)}$$

defines, for all **incoming interfaces** and packet **headers**, **outgoing interfaces** together with **modified headers**.

# Routing

Packet routing sequence can be represented using **sequence of tuples**:

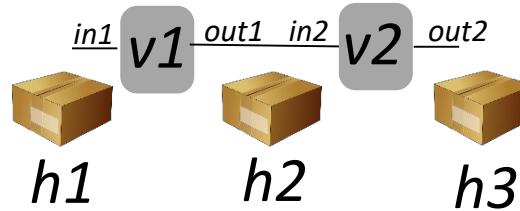


- Example: **routing** (in)finite sequence of tuples

$$(v_1, in_1, h_1, out_1, h_2, F_1),$$

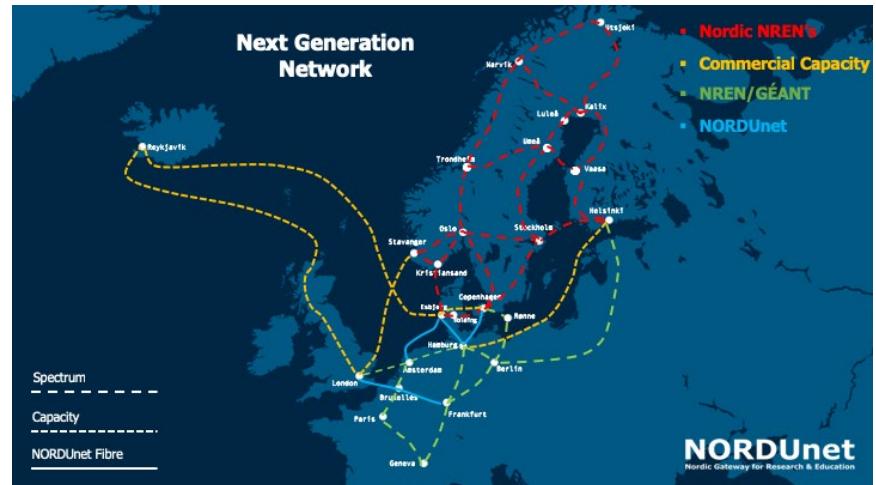
$$(v_2, in_2, h_2, out_2, h_3, F_2),$$

...



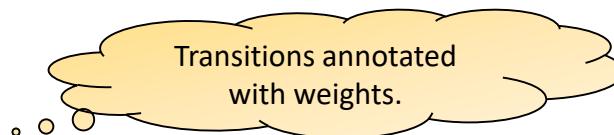
# Case Study: NORDUnet

- Regional service provider
- **24 MPLS routers** geographically distributed across several countries
- Running **Juniper** operating system
- More than 30,000 labels
- Ca. **1 million** forwarding rules in our model
- For most queries of operators: answer ***within seconds***



# Generalizes to Quantitative Properties

- AalWiNes can also be used to test **quantitative properties**
- If query is satisfied, find trace that minimizes:
  - **Hops**
  - Latency (based on a latency value per link)
  - Tunnels



- Approach: **weighted** pushdown automata
  - Fast *poly-time algorithms* exist also for weighted pushdown automata (area of dataflow analysis)
  - Indeed, experiments show: *acceptable overhead* of weighted (quantitative) analysis

# Roadmap

- **A Static Problem:** Policy Compliance  
*Under Failures*
  - AalWiNes: Fast Automated **What-if Analysis** for Networks (INFOCOM 2018, ACM CoNEXT 2018, ACM CoNEXT 2019, TACAS 2021)
- **A Dynamic Problem:** **Scheduling** Consistent Network *Updates*
  - Latte and quantitative extensions (PODC 2015, ICALP 2018, PERFORMANCE 2021)



# More Adaptable Networks

- Software-defined networking also enables networks to be more **adaptable**
- Attractive for:
  - Fine-grained *traffic engineering* (e.g., at Google)
  - Accounting for changes in the demand  
(*spatio-temporal structure*)
  - Security policy changes
  - Service relocation
  - *Maintenance* work
  - Link/node failures
  - ...

COMMUNICATIONS  
OF THE  
ACM

HOME CURRENT ISSUE NEWS BLOGS OPINION RESEARCH PRACTICE CAREERS ARCHIVE VIDEOS

Home / Magazine Archive / March 2016 (Vol. 59, No. 3) / A Purpose-Built Global Network: Google's Move to SDN / Full Text

PRACTICE

## A Purpose-Built Global Network: Google's Move to SDN

Communications of the ACM, March 2016, Vol. 59 No. 3, Pages 46-54  
10.1145/2814326  
Comments

VIEW AS: SHARE:

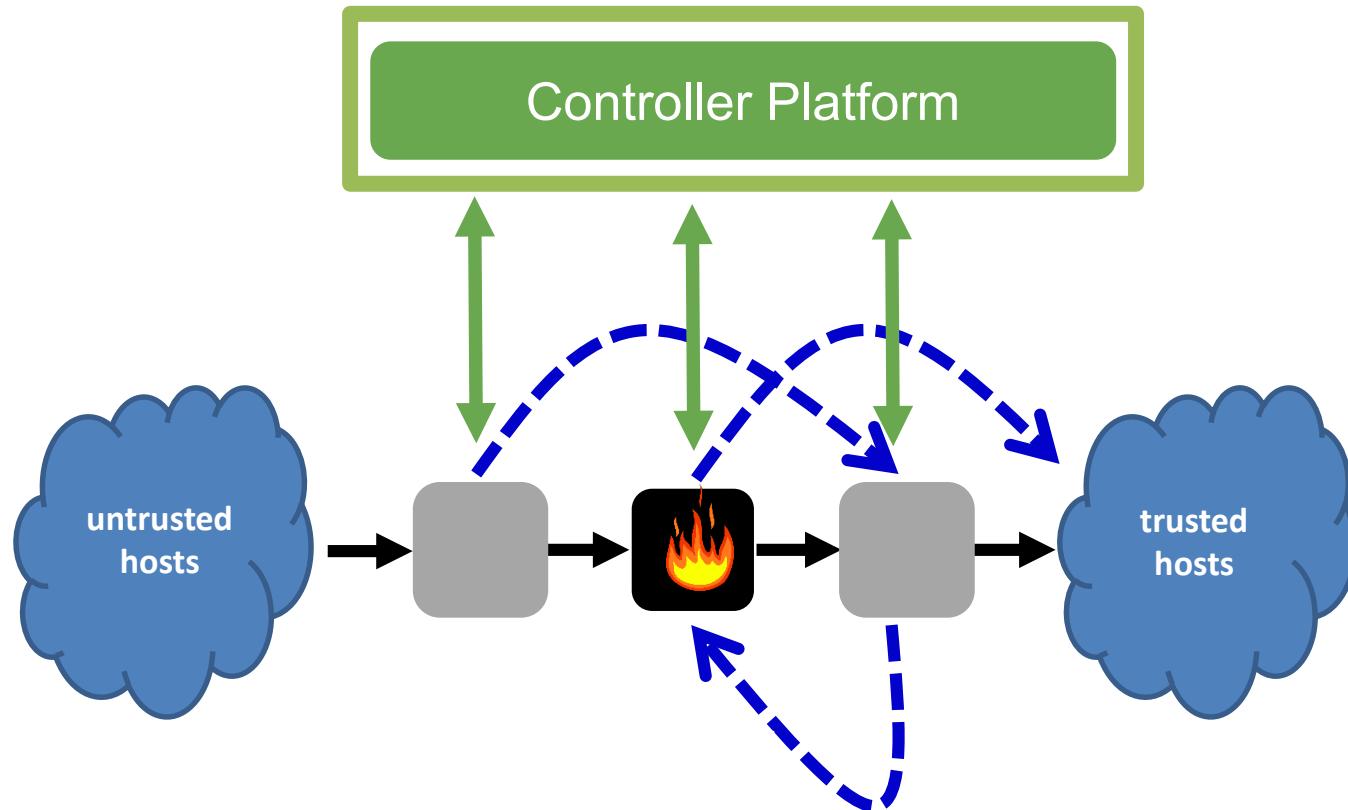


Everything about Google is at scale, of course market cap of legendary proportions, an unrivaled talent pool, enough intellectual property to keep armies of attorneys in Guelts for life, and, oh yeah, a private WAN bigger than you can possibly imagine that also happens to be growing substantially faster than the Internet as a whole.

SIGN IN for Full Access  
User Name  
Password  
» Forget Password?  
» Create an ACM Web Account  
SIGN IN

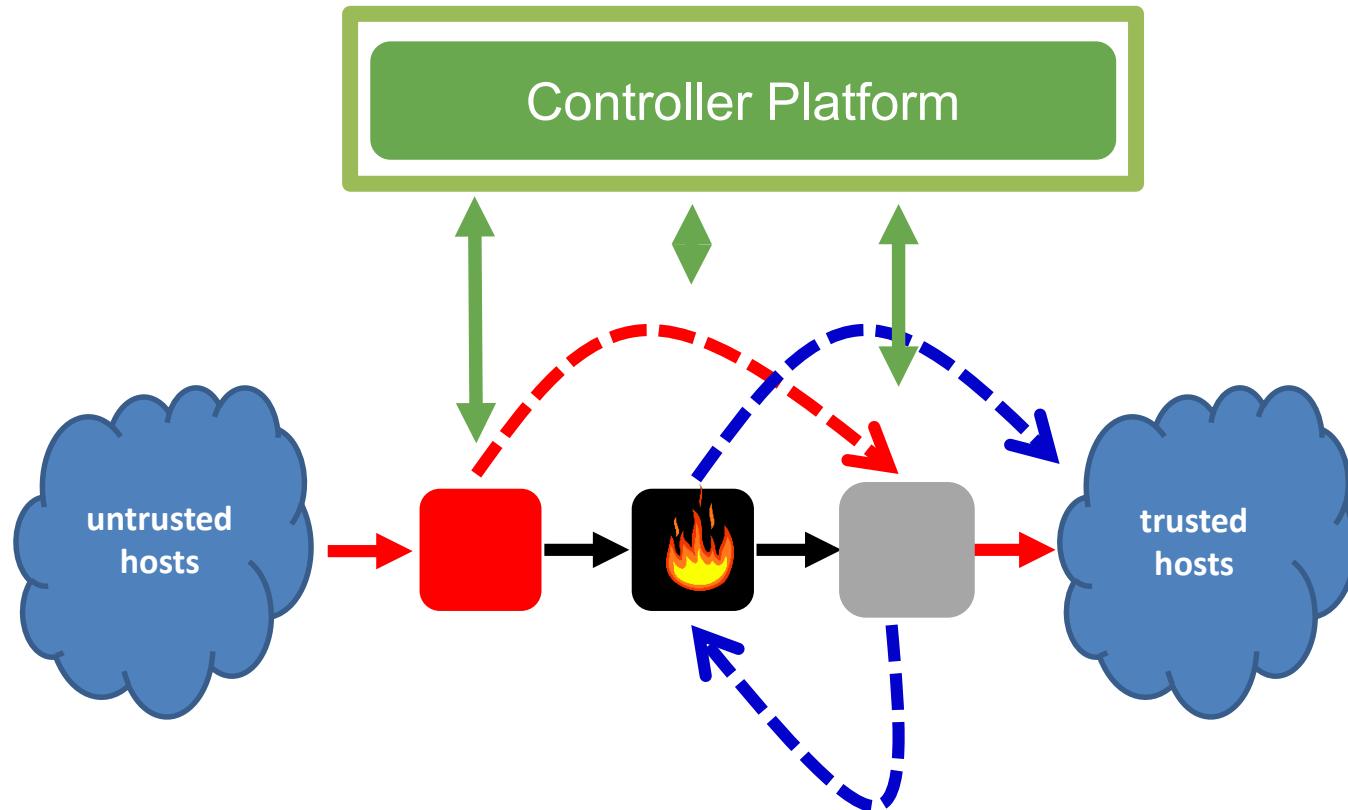
ARTICLE CONTENTS:  
Article

# Introduces a New Challenge: Consistent Update



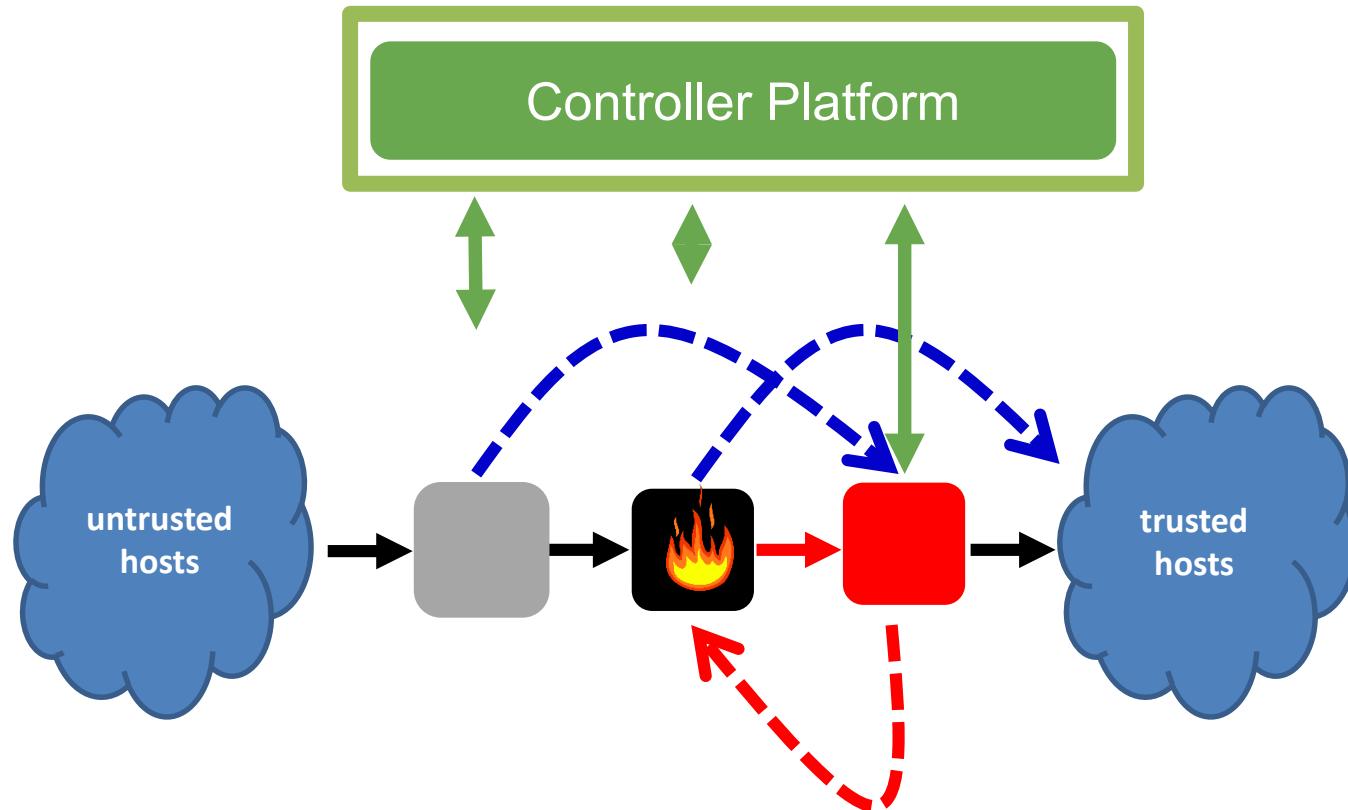
**Invariant:** Traffic from untrusted hosts to trusted hosts via **firewall!**

# Introduces a New Challenge: Consistent Update



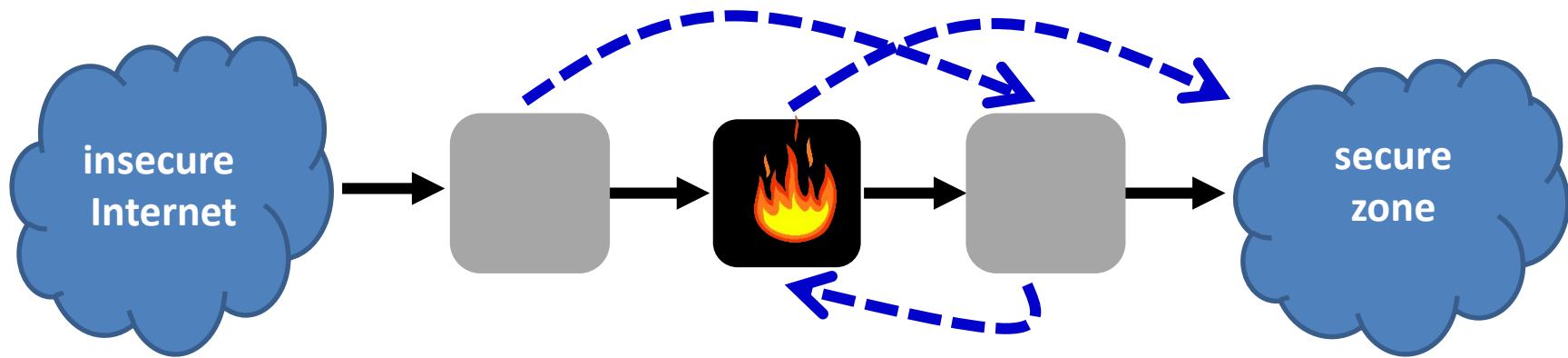
**Invariant:** Traffic from untrusted hosts to trusted hosts via **firewall!**

# Introduces a New Challenge: Consistent Update



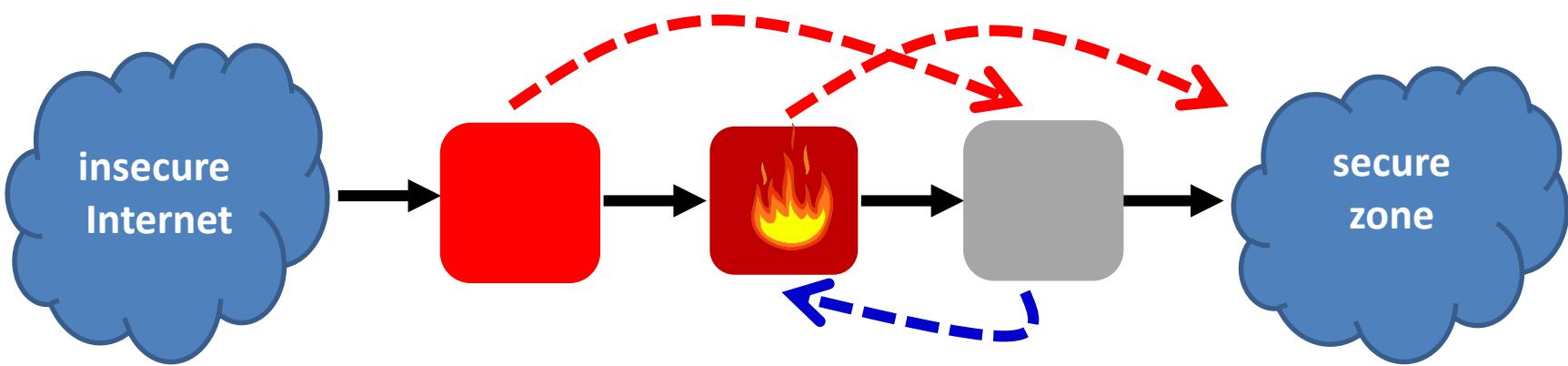
**Invariant:** Traffic from untrusted hosts to trusted hosts via **firewall!**

# Question: How To Update Loop-Free?

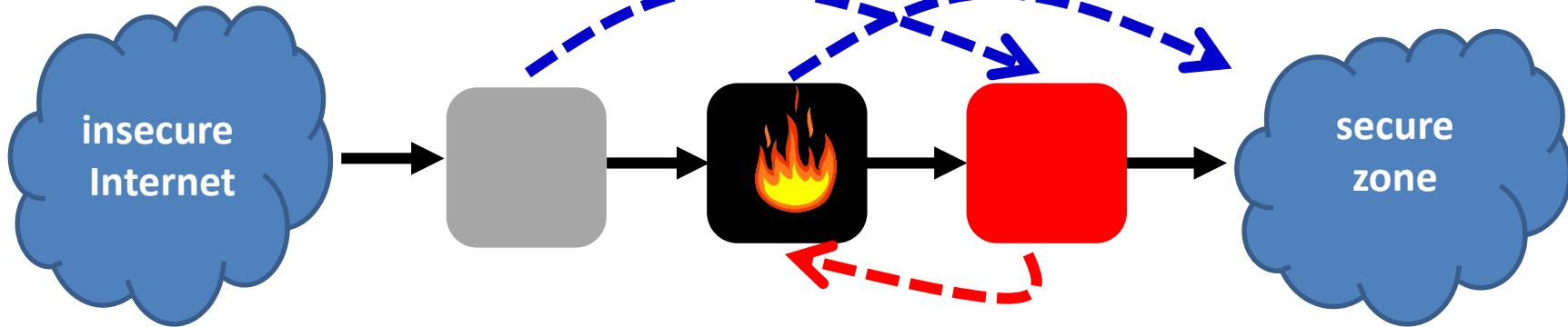


# In 2 Rounds!

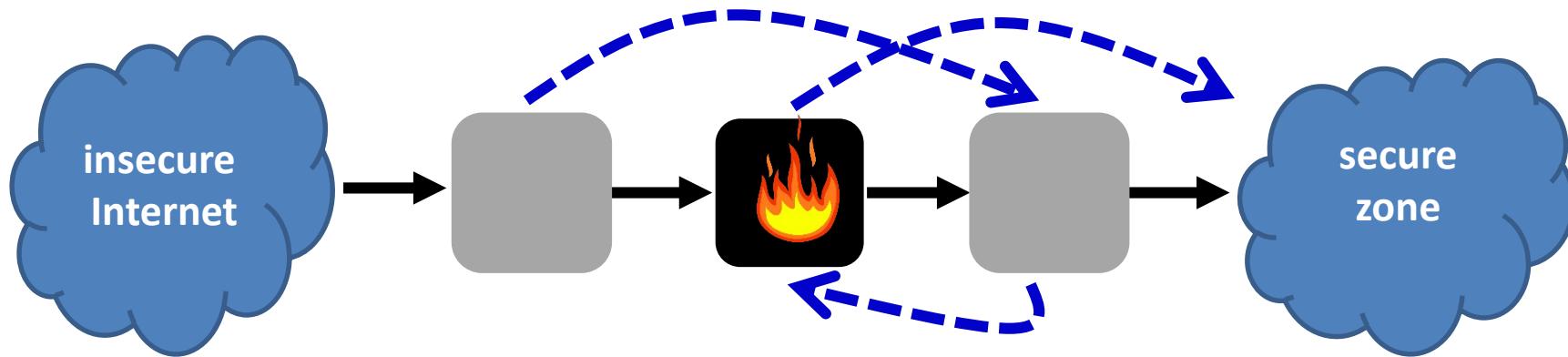
R1:



R2:

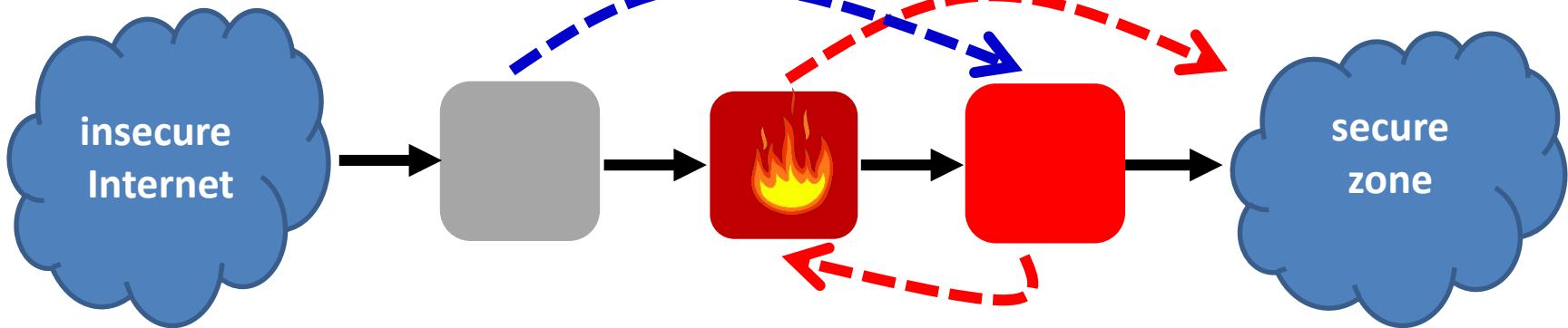


# Background: How To Enforce Waypoint?

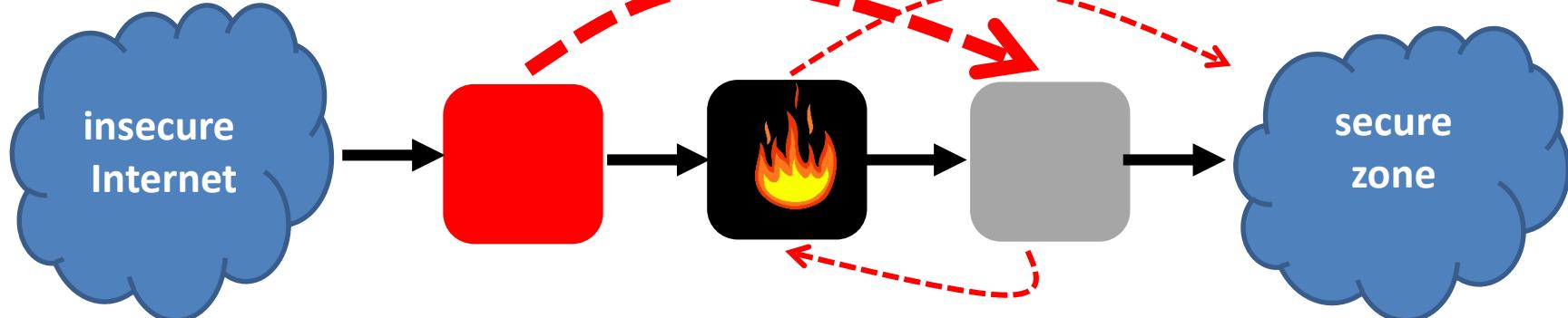


# In 2 Rounds!

R1:

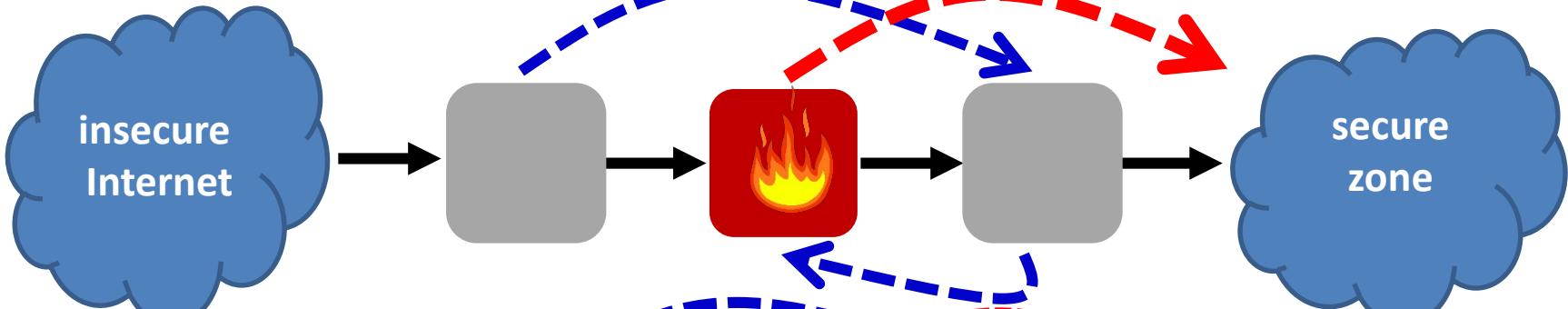


R2:

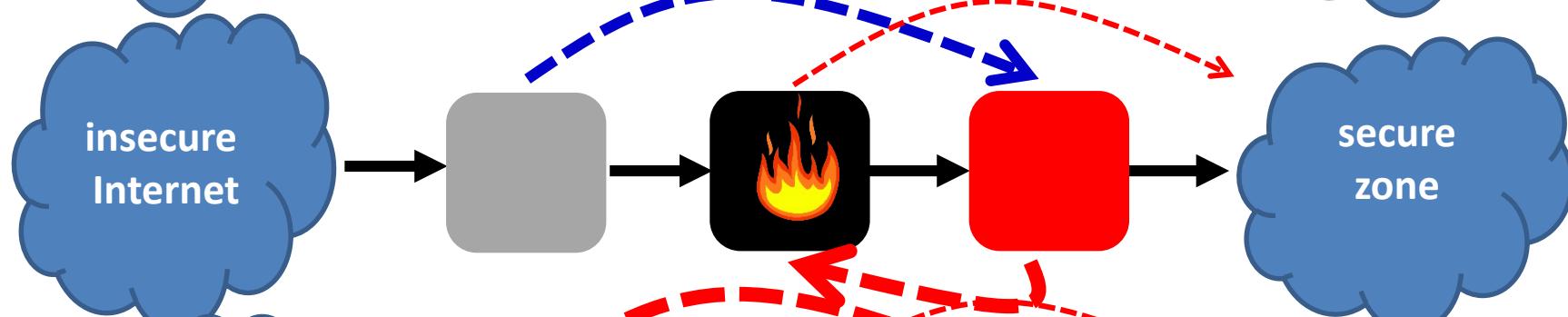


Loop-Free and Waypoint?  
3 Rounds!

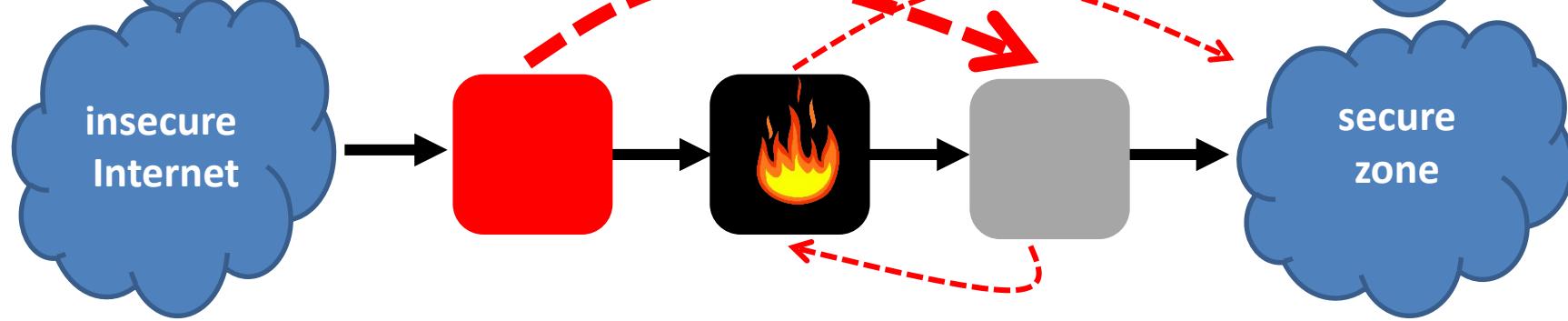
R1:



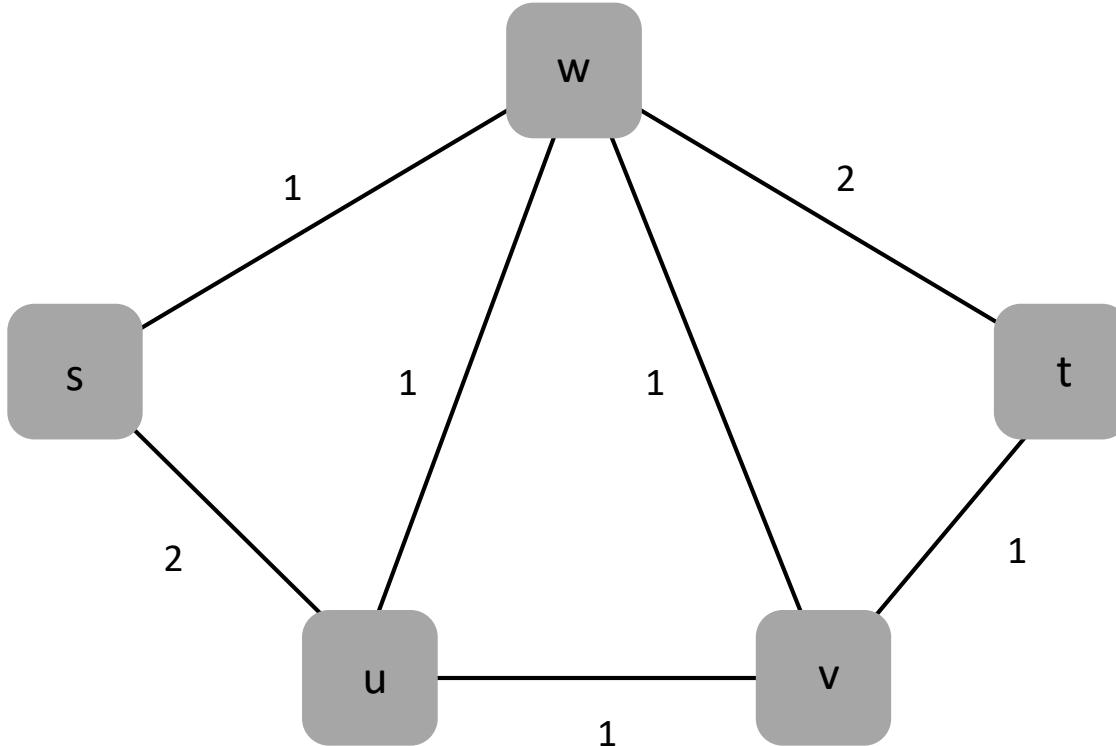
R2:



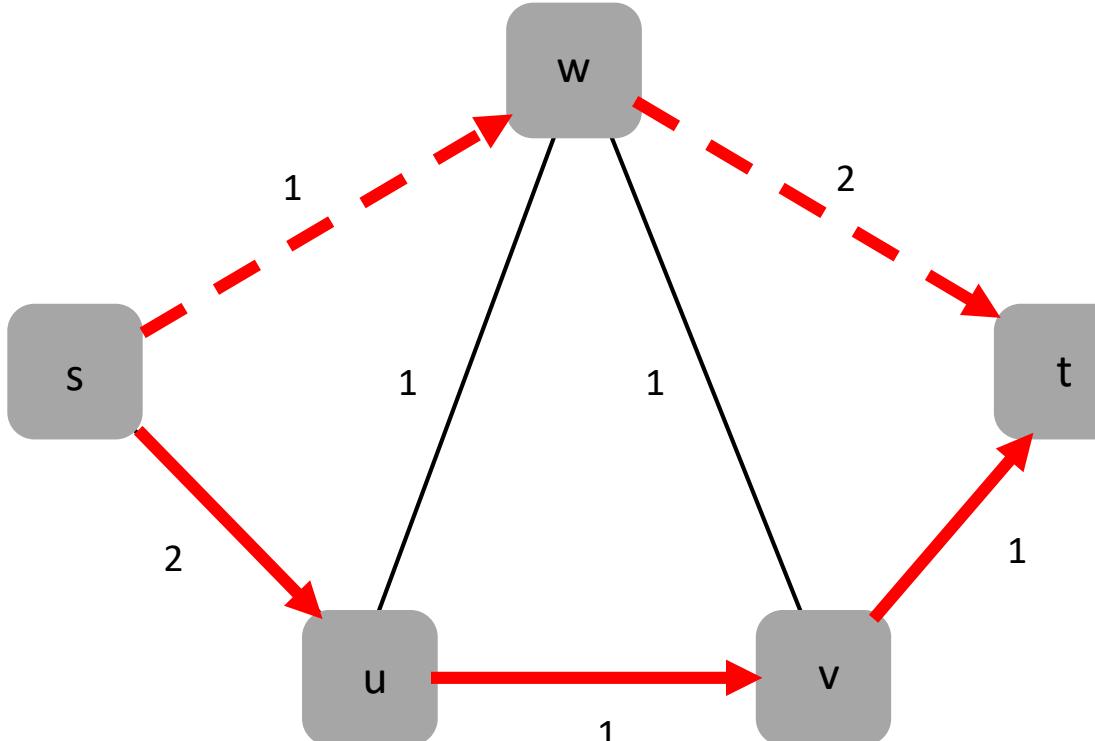
R3:



# Accounting for Quantitative Aspects

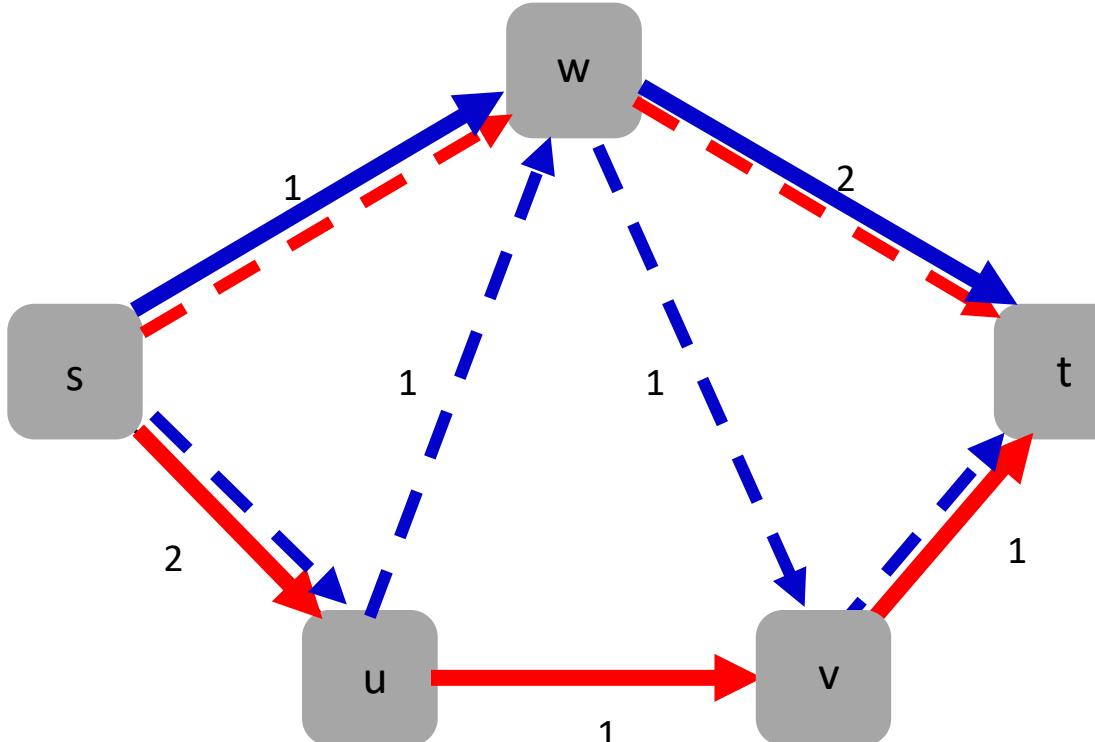


# Accounting for Quantitative Aspects



Flow 1

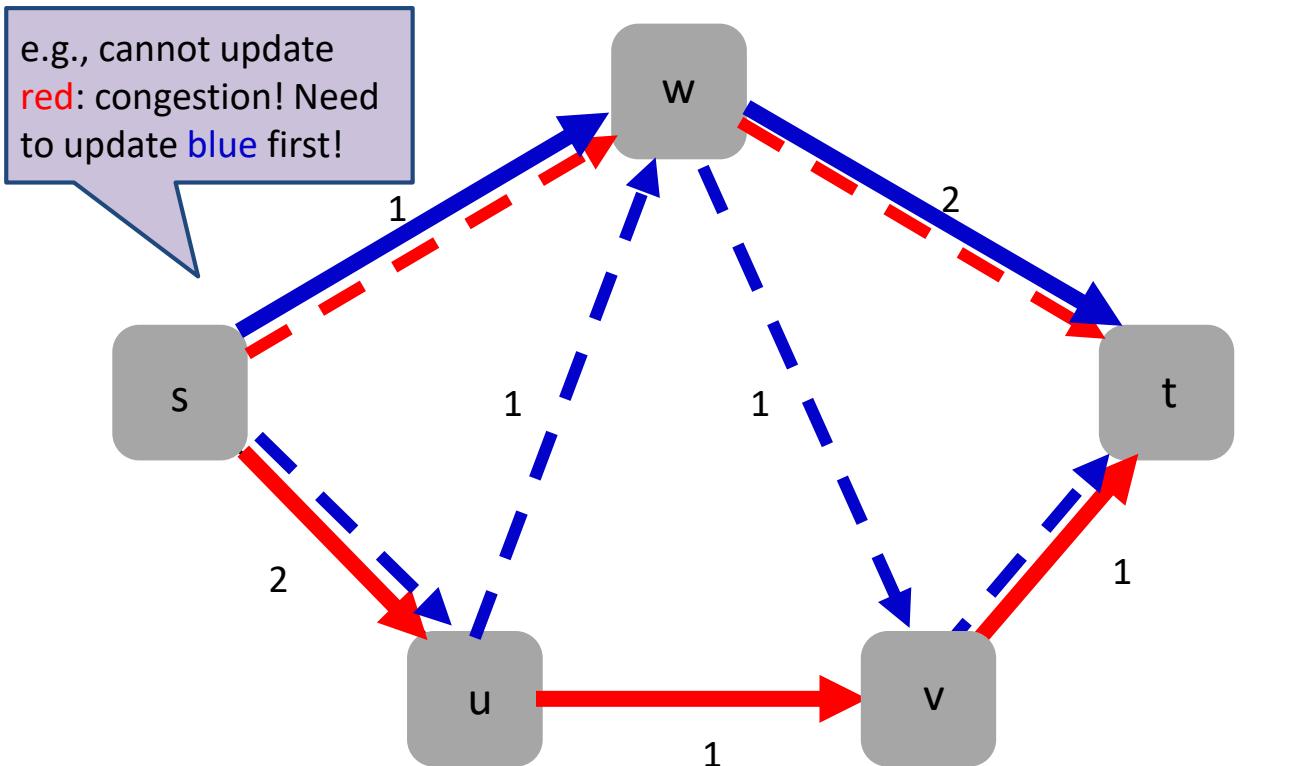
# Accounting for Quantitative Aspects



Can you find an update schedule?

Flow 1  
Flow 2

# Accounting for Quantitative Aspects

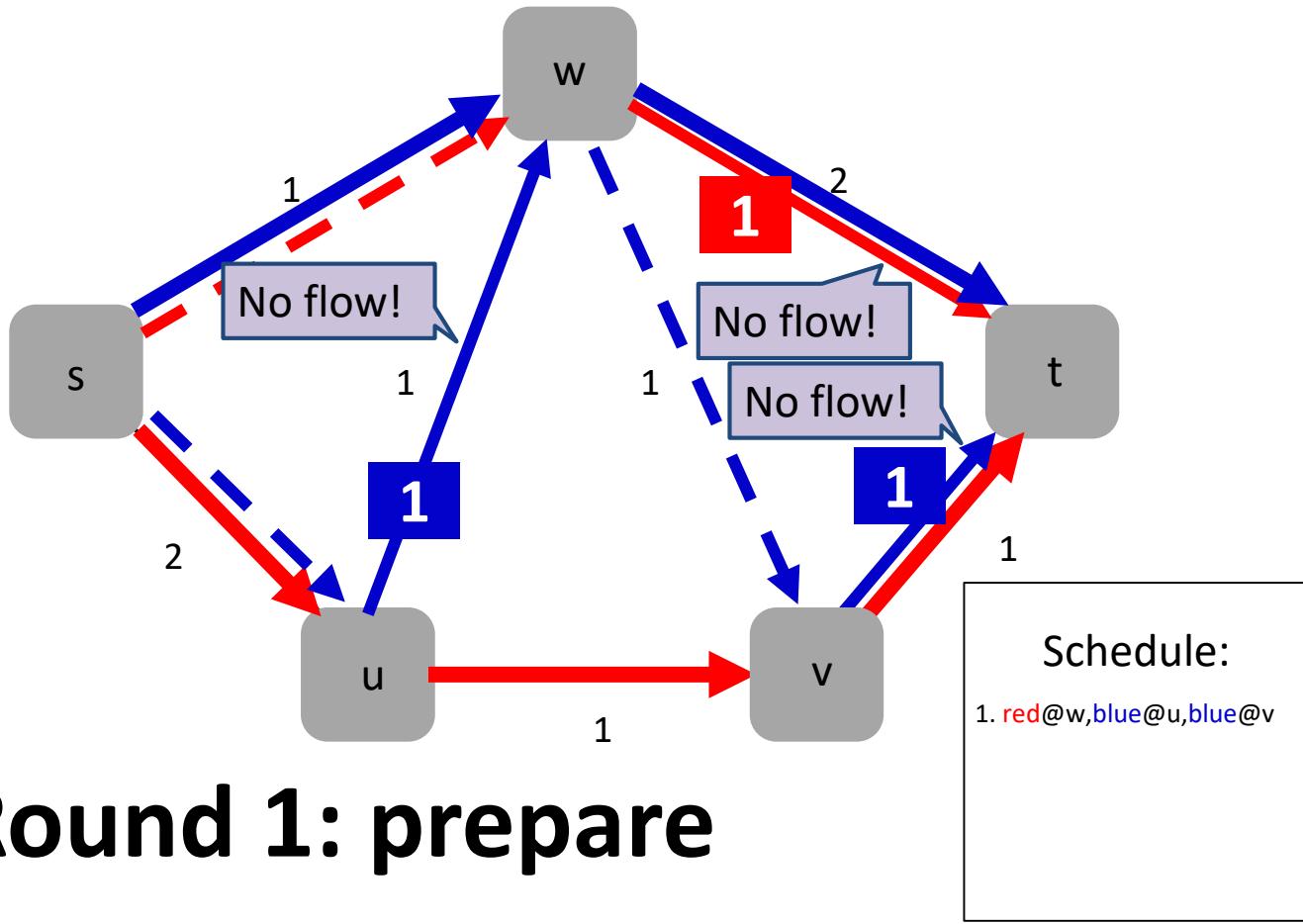


Can you find an update schedule?

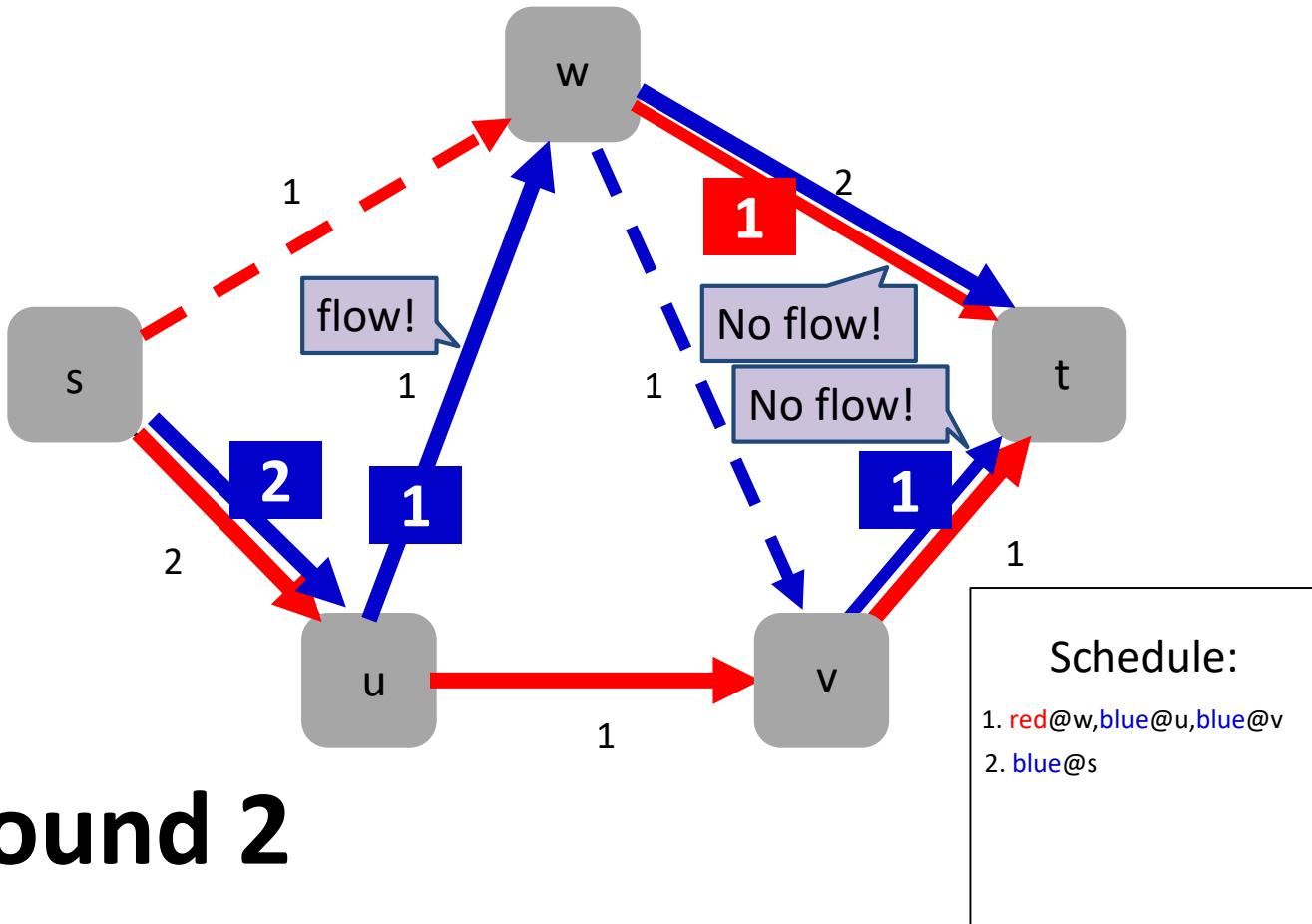
Flow 1

Flow 2

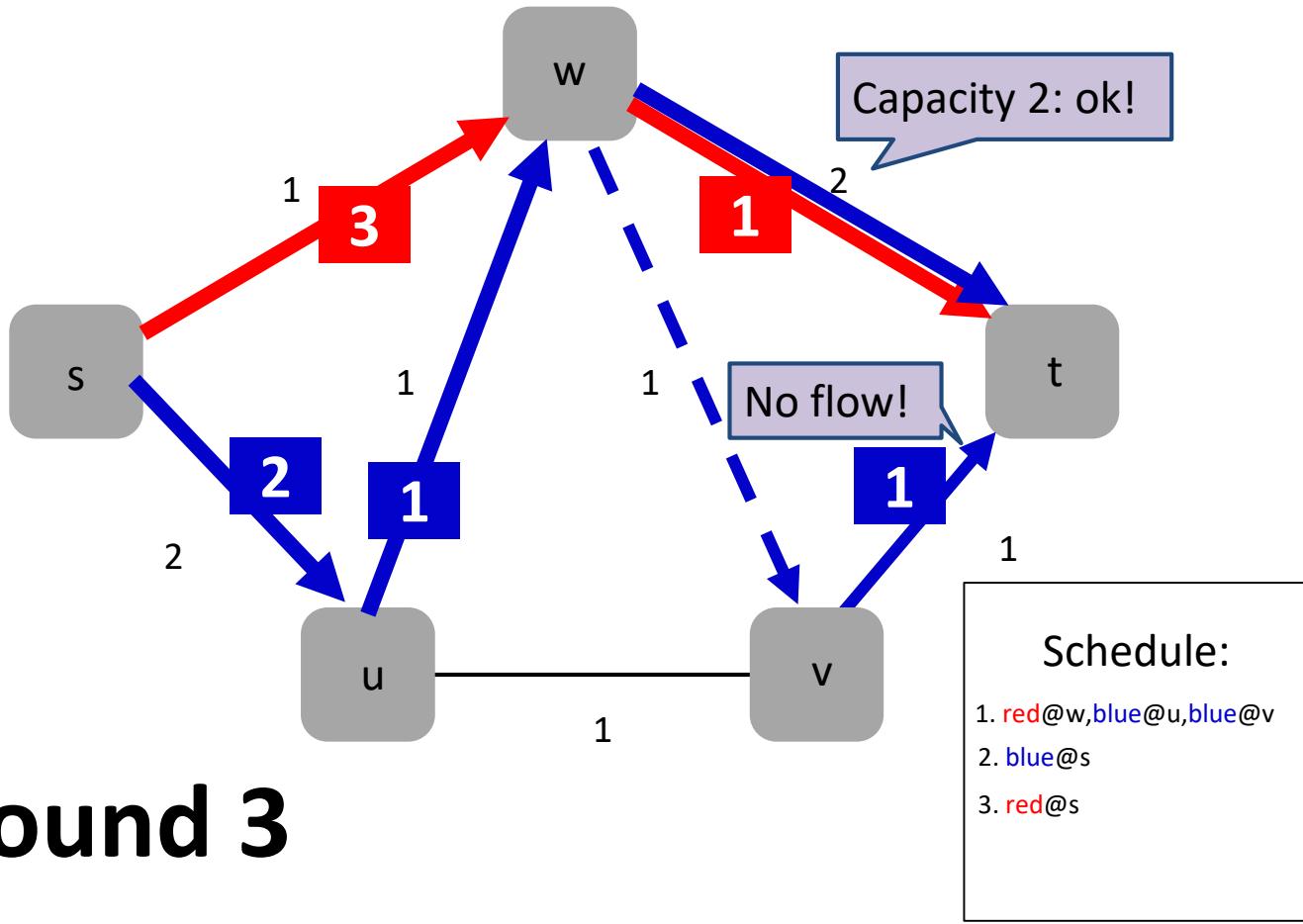
# Accounting for Quantitative Aspects



# Accounting for Quantitative Aspects

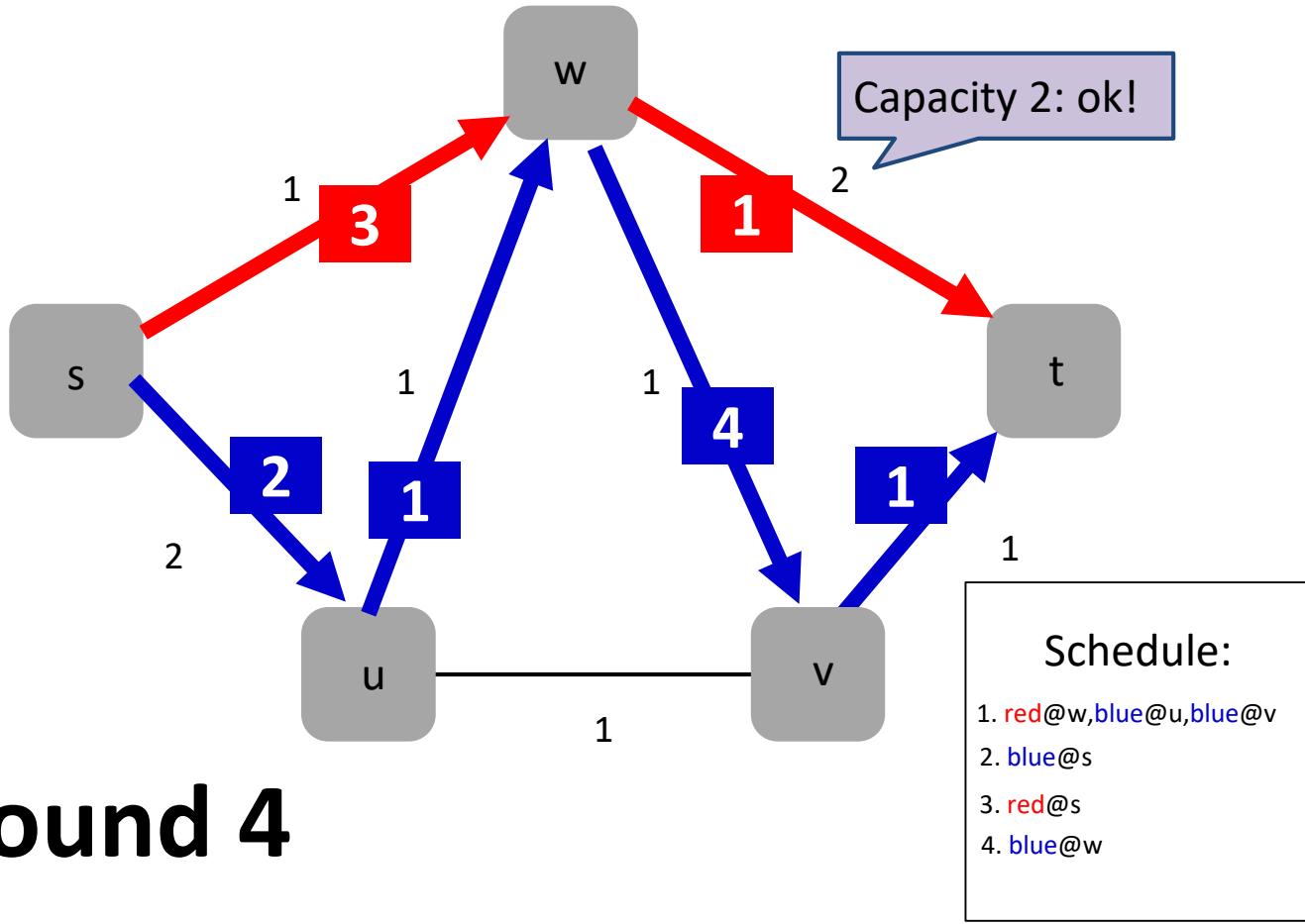


# Accounting for Quantitative Aspects

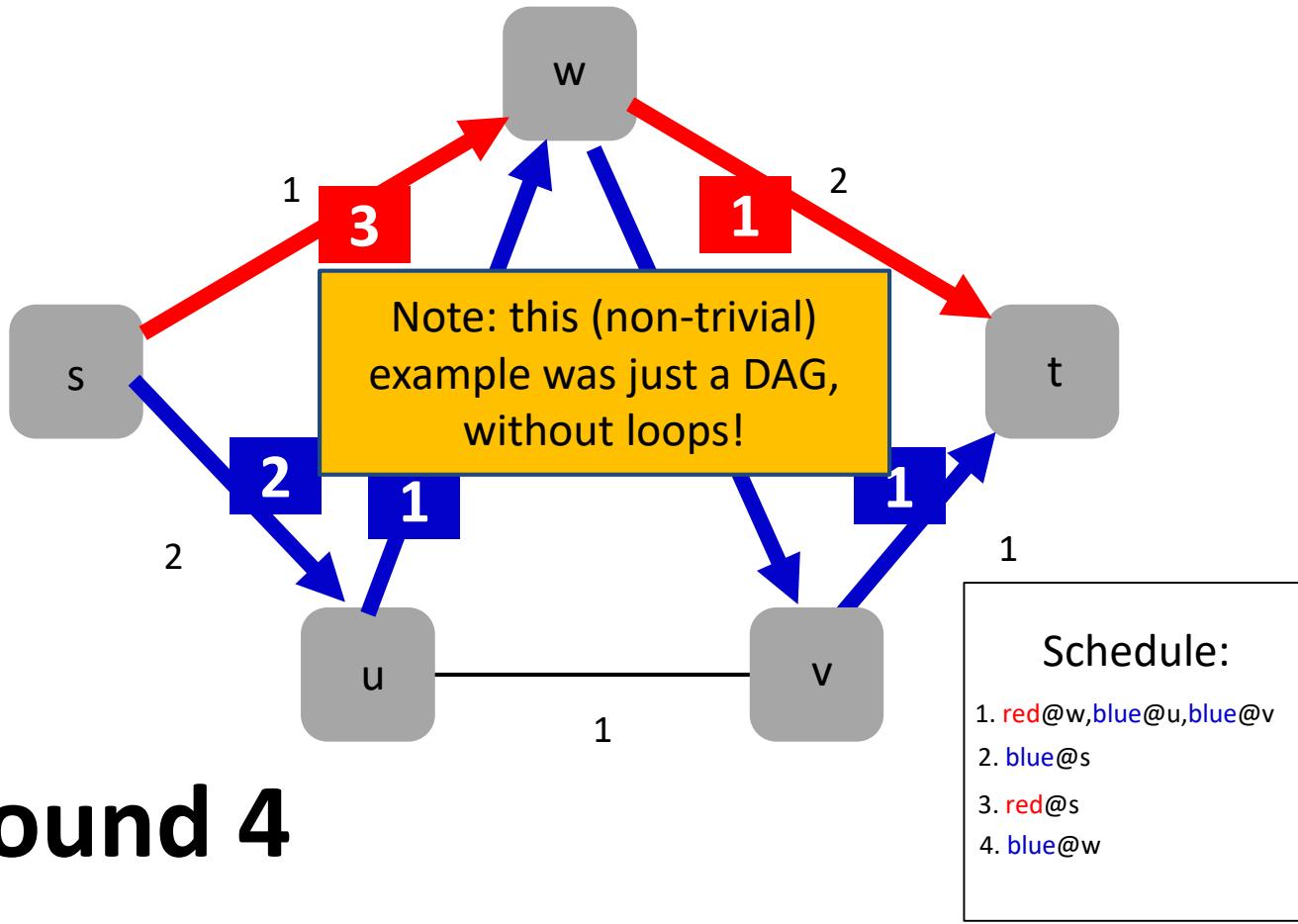


Round 3

# Accounting for Quantitative Aspects



# Accounting for Quantitative Aspects



## Round 4

# Latte: Shortest Consistent Update Schedules

- A first approach: **fast updates** by accounting for temporal properties
  - E.g., different packet types have different *processing times*
  - Requires a fixed *update order* (e.g., produced by NetSynth)
  - Limited to loop-freedom and waypoint enforcement, and scheduling latency (no congestion)
- Based on **petri nets**: powerful modeling language for distributed systems
  - Configurations: tokens located at places
- Our extension: **Timed-Arc Colored Petri Nets (TACPN)**
  - **Tokens** also contain: *color* information (e.g., different packet *types*) and time information (e.g., modeling *age*)
  - **Places** and input arcs have *time constraints* for each color

# Example: Encoding Network Updates in TACPNs

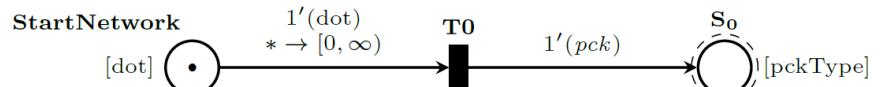
① Gadget to inject packets:

Packets can be of  
different types  
(timings): colors

Color types:  
 $pckType$  is  $\{ VPN, SSH, VoIP \}$   
dot is  $\{\bullet\}$

Variables:  
 $pck$  is  $pckType$

Constants:  
 $vpnMax = 3, sshMax = 1, voipMax = 3$



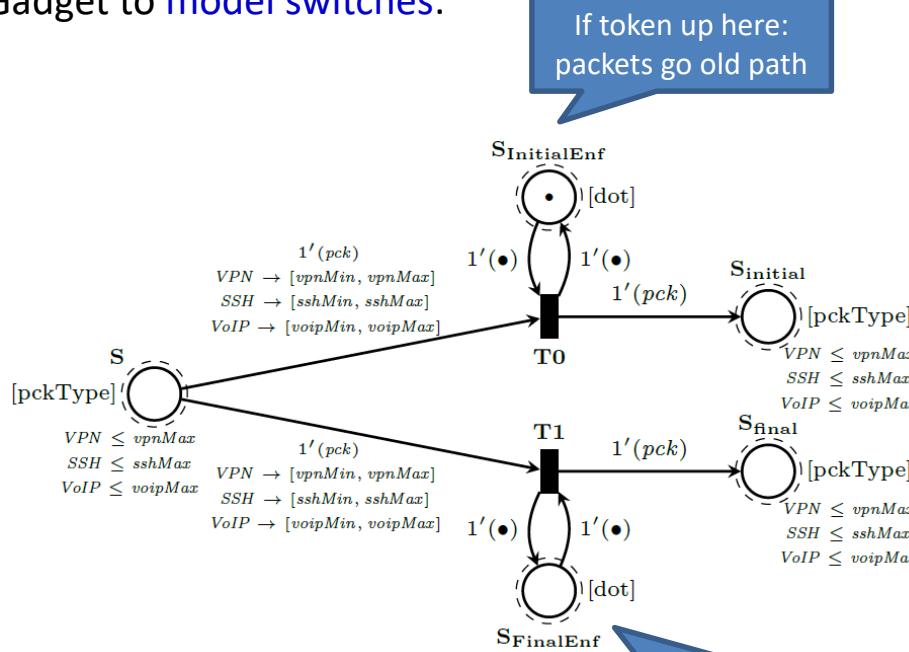
Initially: token at  
this place

Jump to place  $S_0$  and  
generate packet of  
arbitrary type

$$\begin{aligned} VPN &\leq vpnMax \\ SSH &\leq sshMax \\ VoIP &\leq voipMax \end{aligned}$$

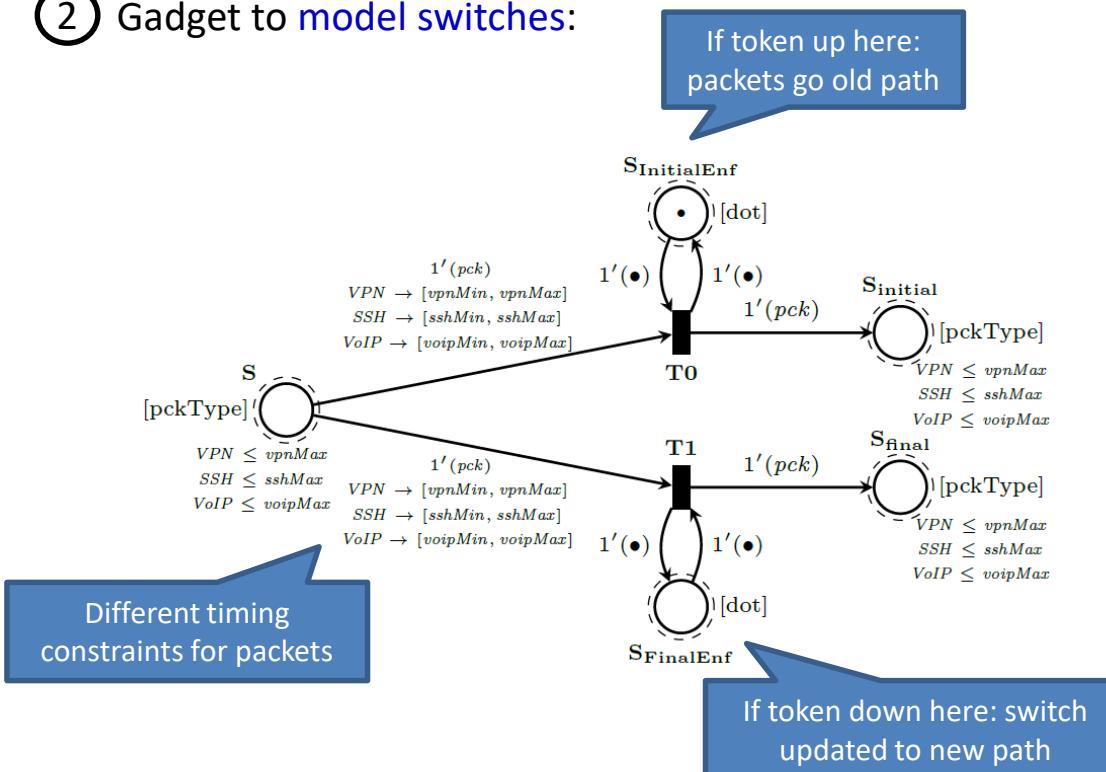
# Example: Encoding Network Updates in TACPNs

② Gadget to model switches:



# Example: Encoding Network Updates in TACPNs

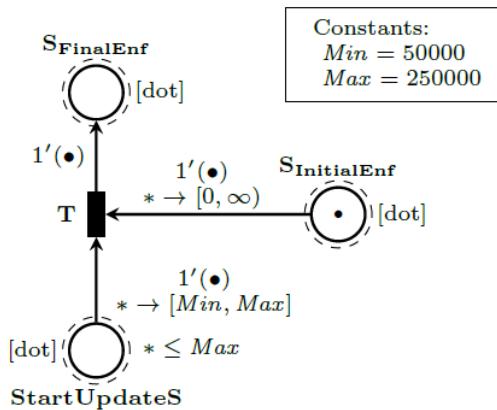
② Gadget to model switches:



# Example: Encoding Network Updates in TACPNs

③ Gadget to model switch update:

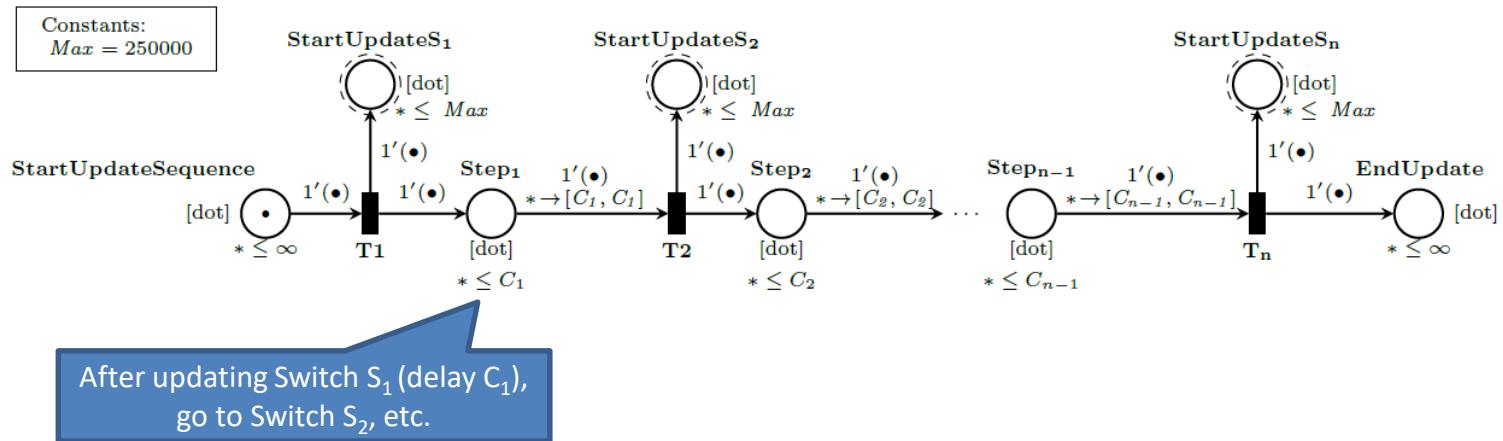
How to change between initial and final switch configuration



Starting here, the update can take time between min and max

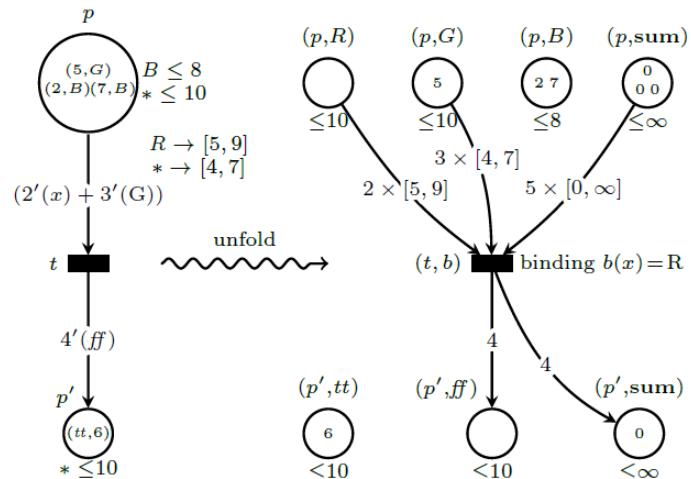
# Example: Encoding Network Updates in TACPNs

- ④ Connecting the pieces: initialization of update sequence for all  $n$  switches



# Analysis

The constructed nets can be analyzed efficiently via their *unfolding* into existing *timed-arc Petri nets*.



Preserves bisimilarity!

# Improved Latency of Update Schedules

Network	Route length	Verification time [s]	Default update time [s]	Optimized update time [s]	Improvement [%]
<i>TLex</i>	4	0.74	3.58	0.25	92.30%
<i>HiberniaIreland</i>	5	1.02	6.05	0.28	95.50%
<i>Harnet</i>	6	1.42	9.08	0.28	96.97%
<i>UniC</i>	7	1.49	12.65	0.28	97.83%
<i>Oxford</i>	8	2.02	16.78	0.28	98.36%
<i>Xeex</i>	10	5.86	26.68	0.28	98.97%
<i>Sunet</i>	11	10.23	32.45	0.28	99.15%
<i>SwitchL3</i>	12	18.88	38.78	0.28	99.29%
<i>Psinet</i>	14	89.67	53.01	0.28	99.48%
<i>Uunet</i>	15	211.86	61.05	0.28	99.55%
<i>Renater2010</i>	16	480.52	69.58	0.28	99.60%
<i>Missouri</i>	25	timeout	171.05	67.10	60.77%
<i>Syringa</i>	35	timeout	336.05	295.35	12.11%
<i>Vt1Wavenet2011</i>	35	timeout	336.06	295.35	12.11%

- Network topologies from the Topology Zoo
- Experiments run on a 64-bit Ubuntu 18.04 laptop

# Improved Latency of Update Schedules

Compared to conservative delays as produced by NetSynth: over 90% improvement.

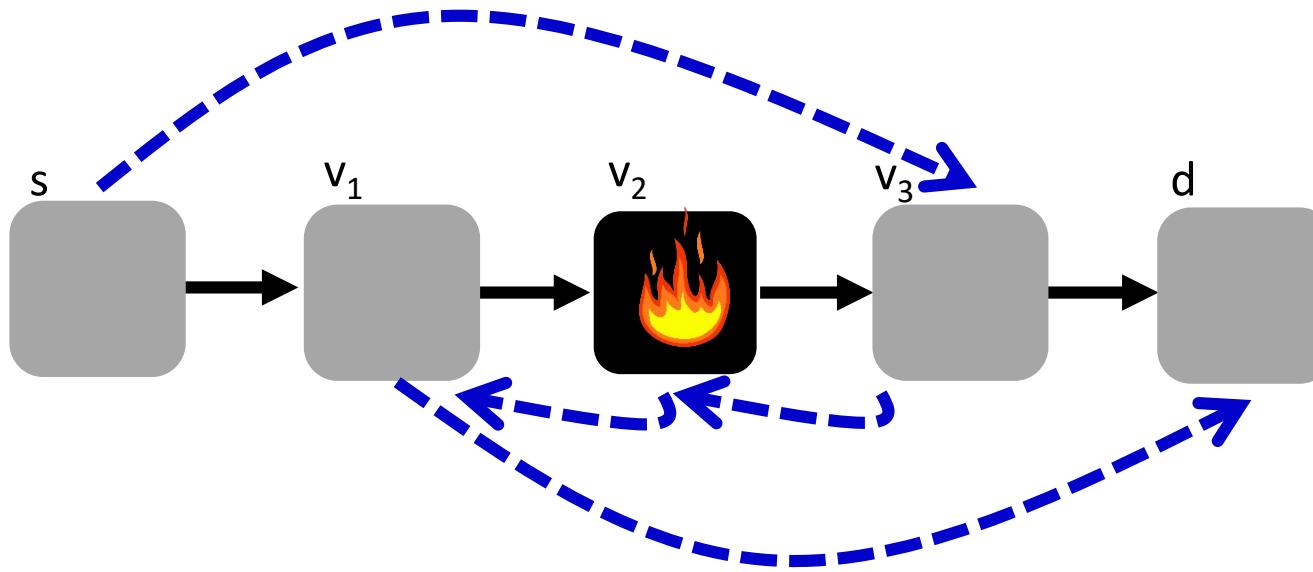
Network	Route length	Verification time [s]	Default update time [s]	Optimized update time [s]	Improvement [%]
<i>TLex</i>	4	0.74	3.58	0.25	92.30%
<i>HiberniaIreland</i>	5	1.02	6.05	0.28	95.50%
<i>Harnet</i>	6	1.42	9.08	0.28	96.97%
<i>UniC</i>	7	1.49	12.65	0.28	97.83%
<i>Oxford</i>	8	2.02	16.78	0.28	98.36%
<i>Xeex</i>	10	5.86	26.68	0.28	98.97%
<i>Sunet</i>	11	10.23	32.45	0.28	99.15%
<i>Cloudflare</i>	12	18.88	38.78	0.28	99.29%
		89.67	53.01	0.28	99.48%
		211.86	61.05	0.28	99.55%
		480.52	69.58	0.28	99.60%
<i>Missouri</i>	25	timeout	171.05	67.10	60.77%
<i>Syringa</i>	35	timeout	336.05	295.35	12.11%
<i>Vt1Wavenet2011</i>	35	timeout	226.06	295.35	12.11%

Up to route length 16, optimal update time can be computed.

Too many updates can be performed concurrently: could be tackled with static analysis (future work).

- Network topologies from the TopoBank
- Experiments run on a 64-bit Ubuntu 18.04 laptop

# Support Beyond „Simple Solutions“



- No loop-free solution with waypoint: cannot update any edge
- But could **first update s to  $v_2$** , then  $v_1, v_2, v_3$ , and finally s again to  $v_3$

# Conclusion

- Finally: networks are moving from manual to **more automated** operations
- Supported by emerging **programmable networks** and their solid theoretical **foundations** and languages
- **Automata-theoretical** approaches can be used to perform fast what-if analysis of the policy compliance (e.g., P-Rex, *AalWiNes*, etc.)
- More **adaptive** network operations further require tools for consistent network update scheduling (e.g., *Latte*, *QSynth*)

Efficient solutions to automatically verify and improve (synthesize) network configurations perhaps **#1 open research challenge** in networking.

- E.g., control plane verification and hybrid, complex network functions (IDS), quantitative aspects, performance aspects and scalability...

Hence looking for collaborations.

# Further Reading

## The AalWines project

<https://aalwines.cs.aau.dk/>

The AalWines network verification suite performs fast (polynomial time) verification of forwarding rules in MPLS routing tables in the presence of link failures. In MPLS, packet labels can be nested in order to provide tunneling through the network or to handle link failures by fast-reroute mechanisms. This mechanism relies on pushing a new MPLS label on top of the label stack, to redirect the flow to go around the failed link. This mechanism can be applied several times in case of multiple link failures. The verification tool allows users to verify whether a given set of rules allow to correctly forward traffic over a wide range of important network properties in polynomial time, parameterized by the maximum number of assumed link failures. At the core of the tool lies an expressive query language for reachability analysis, based on regular expressions, both to specify packet headers as well as constraints on the routes. Specifically, queries are of the form:

< a > b < c < k

## TAPAAL.net

**TAPAAL: Tool for Verification of Timed-Arc Petri Nets**

TAPAAL is a tool for modeling, simulation and verification of Timed-Arc Petri nets (a commonly used graphical model of distributed computations introduced by Carl Adam Petri in 1962). The main idea behind TAPAAL is to support the explicit modeling of real-time, which is associated with the tokens in the net. This is done by adding time annotations to the places and arcs, namely time intervals that restrict the age of tokens that can be used in order to fire the respective transition. In TAPAAL, tool a further extension of this model with age constraints on tokens. This means that tokens have to be at least a certain age before they can be used to fire a transition. This is useful for example previously considered road-arcs) and with inhibitory arcs is implemented.

The TAPAAL tool offers a graphical editor for drawing TAPN models, simulator for experimenting with the designed nets and a verification environment for automatically generating logical formulas formulated in a subset of LTL logic (so-called LTL<sup>ES</sup> AF). As formula verification needs, it also provides the user to check whether a given formula is true or false for a given TAPAAL net. The newest version of TAPAAL is now equipped with three open source verification engines distributed together with TAPAAL (for continuous time semantics, discrete time semantics and a new efficient engine for the verification of untimed nets). Optionally, the user can automatically translate TAPAAL models into UPPAAL and rely on the UPPAAL verification engine.

## Netverify.fun

RESEARCH, NETWORK, VERIFICATION

## Toward Polynomial-Time Verification of Networks with Infinite State Spaces: An Automata-Theoretic Approach



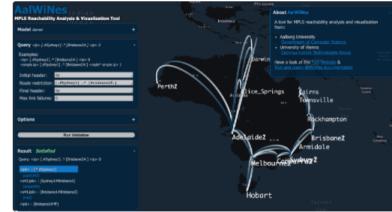
Stefan Schmid [View](#)

Jul 20, 2020 - 6 mins read



Jiri Sriba [View](#)

Jul 20, 2020 - 6 mins read



With the increasing scale of communication networks, failures (e.g. link failures) are becoming the norm rather than the exception. Given the critical role such networks play for our digital society, it is important to

# References

## Resilient Capacity-Aware Routing

Stefan Schmid, Nicolas Schnepf and Jiri Srba.

27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (**TACAS**), Virtual Conference, March 2021.

## AalWiNes: A Fast and Quantitative What-If Analysis Tool for MPLS Networks

Peter Gjøl Jensen, Morten Konggaard, Dan Kristiansen, Stefan Schmid, Bernhard Clemens Schrenk, and Jiri Srba.

16th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Barcelona, Spain, December 2020.

## Latte: Improving the Latency of Transiently Consistent Network Update Schedules

Mark Glavind, Niels Christensen, Jiri Srba, and Stefan Schmid.

38th International Symposium on Computer Performance, Modeling, Measurements and Evaluation (**PERFORMANCE**) and ACM Performance Evaluation Review (**PER**), Milan, Italy, November 2020.

## P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures

Jesper Stenbjerg Jensen, Troels Beck Krogh, Jonas Sand Madsen, Stefan Schmid, Jiri Srba, and Marc Tom Thorgersen.

14th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Heraklion/Crete, Greece, December 2018.

## Congestion-Free Rerouting of Flows on DAGs

Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht.

45th International Colloquium on Automata, Languages, and Programming (**ICALP**), Prague, Czech Republic, July 2018.

## Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks

Stefan Schmid and Jiri Srba.

37th IEEE Conference on Computer Communications (**INFOCOM**), Honolulu, Hawaii, USA, April 2018.



Questions?